25 GAMES FOR YOUR TRS-80™ MODEL 100

BY DAVID D. BUSCH

25 GAMES
FOR YOUR
TRS-80™ MODEL 100

Dedicated to David, Jr. and Michael for evaluating the programs in this book; and to Cathy for providing me with two of the world's finest beta testers.

No. 1698
$15.95

25 GAMES FOR YOUR TRS-80™ MODEL 100

BY DAVID D. BUSCH

TAB TAB BOOKS Inc.
BLUE RIDGE SUMMIT, PA 17214

Some of the material in Chapter 1 originally appeared in *Interface Age* magazine, August, 1983.

FIRST EDITION

FIRST PRINTING

Library of Congress Cataloging in Publication Data

Busch, David D.
25 games for your TRS-80 model 100.

Includes index.
1. Computer games. 2. TRS-80 (computer—Programming.
I. Title. II. Title: Twenty-five games for your TRS-80
model 100.
GV1469.2.B82 1984 794.8′2 83-18094
ISBN 0-8306-0698-X
ISBN 0-8306-1698-5 (pbk.)

Contents

Introduction

Play games on a Radio Shack Model 100? Why not? Game playing has been a time-honored activity for computerists whether hobbyist hacker, home user, scientist, or business executive. Some of the really big computer games, like Adventure and Zork, were available for large systems many years before they were adapted for microcomputers. And who do you think were playing those games—the children of the scientists and programmers?

The TRS-80 Model 100 has been slotted into a business role because it is so well-suited for portable business applications. However, that does not detract from its possibilities as a games machine. It is, after all, just as portable as those hand-held games that sell so well. And, with its memory and 80C85 microprocessor, the Model 100 has more power than any hand-held game to date.

This book has two purposes. First, it will give you, in one compact volume, 25 games for the TRS-80 Model 100. Some are word games. Others test your reflexes. A group use the arrow keys on the Model 100 as a sort of joystick to control screen action in arcade style. Nearly all will run in the unexpanded 8K version. If you have extra memory, several games can be loaded and available for play at any time. But, be sure to leave some room for serious work.

Second, the book teaches some BASIC programming techniques. By typing in each program or by examining the listings the reader will become familiar with some of the tricks used by experienced programmers. Many of the special features of the Model 100, such as sound and the ability to peek at memory to see what is printed on the screen, are explained fully. If you have used another Radio Shack computer, this book will help you learn about some of the differences.

This book can be enjoyed by users of all persuasions. Those with little or no programming experience and no desire to learn programming can type in the programs as they are and enjoy them. Generally, the first part of each chapter explains what the program does and tells how to use it without getting into heavy explanations of GOTOs and so forth. Each program also has brief instructions embedded. So, it is possible to run the games with no more understanding than that of a collie as to how the dog food gets in the little cans. Note, however, that some of the programs will go on forever: you can mystify the audience with Computer Magi, land a parachute, or call out Bingo numbers until your fingers are too feeble to press the keys. Alternatively, you could press the shift and break-pause keys simultaneously—and regain control of your machine.

Those with more of an interest in programming can use this book to sharpen their skills. This is not a beginning BASIC tutorial. It presumes you know what a for-next loop is and what happens when a program line says ON A GOSUB. However, a wealth of tips and tricks are presented for the novice and intermediate programmer. You will learn how to write input routines that reject program-crashing errors by naive games players. There is a clear discussion of discovering what is on the screen of the Model 100 using the peek statement. Many concepts crucial to games writing, such as detecting collisions and dealing a deck of cards, are introduced.

For ease of learning, programs are generally not written differently just for the sake of being different. The programs are not jammed together with complex multistatement lines, no spaces, and lots of else conditionals. Most lines have only one or two statements, except where a series of operations follows a true IF statement.

Variable names in many programs are selected for ease of understanding their purpose—GRAIN, CASH, BET—even though they take up a bit more memory space. Most programs are consistent in use of variables: N is used as a loop counter; DU is a dummy variable; A$ is used in INKEY$ loops for user input; F$ is used as a PRINT USING format. In addition, good programming practices have been adhered to wherever possible. Most of the time, you won't find a variable initialized inside a loop, where it must be done over and over needlessly (although I do see at least one case where this happened).

Moreover, a general top-to-bottom program flow is used, without an excessive reliance on subroutines. For example, A$=INKEY$:IF A$=" " GOTO may be used repeatedly in a program, when a subroutine might have been faster to write. However, repeating the program lines makes the logic a bit easier to follow. Several programs do jump around a bit, and other good programming techniques are temporarily ignored, usually to illustrate how to do something else.

Hardware-shy users can breathe easily, however. Here you'll find no discussion, outside the first chapter, of microprocessors, interfaces, RAMs, or ROMs. The emphasis is on programming, and fun.

Part of my goal in writing this book was to have a good time myself. I've kept a light-hearted tone throughout, and slipped in a few gags here and there. Those of you who are regular readers of magazines aimed at TRS-80 users are probably familiar with my Kitchen Table, Inc. series of humor articles. These 25 games are not KTI products, I assure you. All have been tested on an actual TRS-80 Model 100 and will perform as described.

In keying in these programs, you can safely remove any remarks. These lines were all added after the programs had been written, and so there are no GOTOs or GOSUBs that send control to a remark line. Nearly all remark lines, except the program title block, have been created with line numbers ending in five. I recommend leaving the title block remarks in the program for easy identification later.

If you decide to modify a program, be sure and save a backup copy as originally typed in. Although I try to explain what every module does and point out some of the stranger program lines, it is entirely possible that you do not understand what every line does, and exactly what the effects of changing or deleting a given line will be. It's better to have a backup to return to if a program gets hopelessly muddled.

Chapter 1

Your Model 100

At this writing, the TRS-80 Model 100 has only been on the market a short time, so there are no commercial games available for it. In fact, most of the programmers I know are busy writing business-oriented software, including bisync communications programs and accounting software. However, there are ways to find games for your machine. This book is one way. Reading magazines that are specifically written for the TRS-80 Model 100, such as *Portable 100* (Highland Mill, Camden, ME 04843) will lead you to other games.

However, the best source I can recommend is a user's group made up specifically of Model 100 owners. If you don't have such a local group, you'll find a concentration of Model 100 users beyond your wildest dreams as close as your telephone. I'm referring to the CompuServe Information Service, which has its own Model 100 Special Interest Group. Users in this SIG, which was born within a few weeks of the introduction of the Model 100, have donated a wealth of utility, business, and, yes, games programs. All are available for downloading through TELCOM.

In case you are not familiar with CIS, I'll describe it briefly. CompuServe is an information utility that can be accessed by calling a local telephone number in about 300 cities. A WATS line is available for those beyond reach of local service, although at an extra charge. CompuServe can also be accessed through Tymnet and Telenet.

The basic charge is $6 an hour after six p.m., with a $2 surcharge for Tymnet and Telenet. If you purchased the Radio Shack telephone cable for the Model 100, you received a free membership to CIS, along with one hour of connect time. The menu-driven service is not really confusing for first time users, but you may have overlooked the Model 100 Special Interest Group. It can be accessed from any ! prompt by typing **GO PCS 154**. The Model 100 SIG has no signup fees; your only cost is for the $6 connect time.

Once connected to the SIG, you'll see a bulletin board with several sections. Users may leave messages, ask questions, and leave copies of short programs right on the message board. All manner of Model 100 owners regularly visit the board; as many as 256 messages may be present at one time, and the turnover can be as high as 100 or more messages a day. On the board, I've seen useful peeks and pokes, tips for wiring to difficult telephone lines, many songs that can be played on the Model 100, and some games.

In addition, the SIG has several databases, chock full of programs uploaded by members. Some games are here, as well. You could easily download three or four games in half an hour. What other software costs you $3.00?

SMALL SIZE BRINGS NEW CAPABILITIES

Before I get into the actual games, I want to provide a brief overview of the Model 100 and its capabilities. Most owners will understand their computers to some degree and have a few specific uses for them. They may not have a thorough comprehension, however, of the broad range of capabilities the Model 100 has and its place in the overall microcomputer picture.

I think the Model 100 will be remembered as a landmark personal computer. Landmark products, those that either introduce a concept or make an idea really practical for the first time, can very quickly define a whole new market area. Transportable computers, for example, were very hard to sell before the Osborne I combined portability and bundled software at a reasonable price. Similarly, the Radio Shack TRS-80 Model 100 computer is a landmark device even though units that perform some of the same functions and have similar features have been available from other manufacturers for months.

But, while the first keyboard-sized portable word processing unit, marketed here several years ago by a Japanese firm, cost $1000-plus and could not be used as a computer, the Model 100 will do text manipulation and computing for $799. More recently, another Japanese company has introduced a true computer which costs about the same as the Model 100, but it can't display eight 40-character lines, nor does it have a built-in modem. Another predecessor of the Model 100 displays four 80-character lines and has massive amounts of bubble memory. However, it costs three times as much as the Model 100. Comparisons with existing products are easy, but the point is that Radio Shack and the Japanese company that produces the computer have managed to make the briefcase computer practical for serious use for the very first time.

The Model 100 weighs less than four pounds, and is about the size of a two-inch thick notebook. Based on the 8085 microprocessor chip, it boasts a typewriter keyboard, a 40 column, eight line liquid crystal display, and a built-in 300 baud direct connect modem. Both RS-232 and parallel printer interfaces are standard, and the 8K of nonvolatile CMOS memory can be expanded to 32K in 8K increments.

The unit uses both nonremovable rechargeable batteries and four alkaline AA cells. Radio Shack notes that the Model 100 can be operated one hour a day for about 20 days before the batteries have to be replaced. User programs and data are preserved when the computer is turned off.

The 32K of ROM includes a very complete extended Microsoft BASIC and several applications programs, including a surprisingly flexible word processing module, and telecommunications software.

The Model 100 is aimed squarely at those who need the utmost in portability, but don't want to sacrifice computing power. It will do things that are either impossible or inconvenient to do with both pocket computers and sewing-machine sized transportables.

When the Model 100 is turned on, the user is shown a menu of program choices, unless the owner has elected to use the automatic run feature to power up directly into a program. These include TELCOM, the communications program; BASIC; TEXT, the word processing program; an appointment scheduler, and an address program. User entered programs will also appear on the menu. Any of these can be run simply by positioning the cursor over the program name with the arrow keys or by typing the name on the keyboard. When the ENTER key is pressed, the program commences.

The Model 100 is possibly the first tiny portable that is entirely practical for word-processing on the go. The full-sized keyboard has a good feel and will be comfortable for most touch typists. I prefer to tuck a checkbook or similarly sized prop under the unit when typing on a table or desk. This is not to change the viewing angle: that can be adjusted by a thumbwheel on the right side of the computer. I find that slightly elevating the front of the computer provides a more comfortable typing angle. However, I can and have used the computer happily in my lap at airport boarding areas or while watching television at home.

The eight line display is probably the minimum size that can be used comfortably for word processing. Single line displays are difficult to use for editing, I have found. On the Model 100, the screen is generally fairly easy to read, although it can become difficult to see in dim light. The scrolling of lines is more jerky than CRT users are accustomed to.

The word processing program makes extensive use of special function keys that are arrayed along the top of the keyboard. F1 through F8 can be used to search for a specific string, load or save a file, copy text into the text buffer, cut text from the file, select text to be copied or cut, or go back to the main menu.

The features of the eight function keys change from program to program and can be displayed on the screen by pressing an additional label function key. When labels are displayed, only seven lines of text can be seen at a time. Three other function keys have permanent uses. The paste key takes text that has been stored in the text or "paste," buffer and places it where the cursor has been positioned. The print key will send the file to a printer, and the fourth key is a break/pause control.

Arrow keys move the cursor quickly about the text, with the shift and control keys used to enlarge the increment of movement. Control up-arrow, for example, moves the cursor to the beginning of the text file, while shift up-arrow moves it to the top of the current screen.

To move a portion of text into the paste buffer (either to delete it or to move it and paste it somewhere else), the user positions the cursor at the start and presses the F7 (select) key. Then, cursor movements are used to highlight the text, which is shown in reverse printing.

COMMUNICATIONS BUILT IN

The TELCOM program is equally easy to use. When the user presses the find key and enters a name, the computer will search the ADRS data file for a phone number, which can be dialed automatically by pressing the call key. The user can also type in a phone number for autodialing by the Model 100. A standard RJ11 modular plug is inserted in the user's phone jack. Acoustic cups are available for use with other phones. Conceivably, the Model 100 could be used as the world's most expensive automatic phone dialer, as the operator can pick up the phone any time after dialing is completed.

Although the Model 100 is set to use the internal modem, the STAT command can reroute telecommunications through the built-in RS232 interface and an external modem at 300 or 1200 baud, or some other rate. A simple string of characters is entered to change word length, parity, stop bits, XON/XOFF status, and pulse dialing rate (either 10 or 20 pulses per second may be used).

It's difficult to do justice to the sophisticated auto log ons possible with the computer. The "phone numbers" stored in the ADRS data base can include a complete log on sequence, including pauses, and a command to wait for a certain character to be sent from the host before proceeding.

For example, CompuServe expects a control-C to initiate log on and then prompts USER ID: and PASSWORD. A typical string stored in the ADRS file might be:

```
CIS:867-1237:<=[C?U71505,1024[M?
                PSECRET[M>
```

The part within the < > consists of the following: the equals sign, which tells the computer to wait two seconds following connection; a control-C, which is transmitted; a question mark, which causes the computer to wait for the host to send a U (as in USER ID:); the ID number, 71505,1024, which will be sent along with a carriage return (control-M); and a question mark, which will cause the computer to wait for a P before transmitting the password SECRET and another carriage return.

That sequence is greatly simplified. Actually, if you know all the prompts that will be displayed, you can command the computer to log on, query the database for specific information which is downloaded, and then log off—all with little or no operator attention.

Downloading or uploading files is as simple as pressing special function keys. Programs as well as text files can be sent from one computer to another. Using a null modem, it is possible to connect a cable directly from the Model 100 to another computer, such as a TRS-80 Model I/III, and send material that way. No more than 600 baud can be used without losing characters, unless the communications software on the non-Model 100 computer supports the XON/XOFF protocol. I find it so easy to transfer files from one computer to the other that I use my Model I computer as mass storage for Model 100 programs. It's almost like having a nonportable disk drive for the little computer.

FLEXIBLE BASIC INTERPRETER

Few corners have been cut in the BASIC Interpreter. It is a full extended BASIC, fairly compatible with Radio Shack's Level II and Model III BASIC. In fact, I was able to load and run many programs with little or no changes.

BASIC programs that use peeks or pokes will not work, of course, but most others operate quite well. Menus with more than eight lines to display at once cause problems, and the print tab statement can use no more than 40 characters across. The most common change that will be necessary will be to fix RND function. The larger Radio Shack BASICs allow RND arguments larger than 1 (to produce a number in the range 1-7, simply type RND(7)). Model 100's BASIC demands the more complex INT(RND(1)*7)+1 construction.

In all, I counted about 20 commands missing from Model 100 BASIC. Most of these relate to random access file handling (CVD, CVI, CVS, GET, etc.). Only AUTO, FN, and the TRON tracing function are likely to be missed.

Some eight other keywords work with different syntax in Model 100 BASIC. These are CSAVE, CLOAD, EDIT, FRE, LOAD, RND, MERGE, and OPEN.

On the other hand, I was able to count nearly 40 new commands in Model 100 BASIC that are not available on the TRS-80 Model I/III. These range from sound and beep commands, to a files command that returns the computer's menu directory. Commands allow redefining any of the eight special function keys, turning on and off any pixel in the 240 × 64 picture element display, and renaming of files.

Quite clever are the several interrupt routines. Once activated, these direct program control to a specific task, regardless of what else is happening during runtime. One common interrupt routine that most users are familiar with is the ON ERROR . . . GOTO xxx. The on error statement is placed at the beginning of the program. The system interrupts constantly look for an error;

when found, it activates the GOTO portion.

In the Model 100, ON KEY, ON MDM, and ON TIME$ are valid statements, so that a given key can be pressed, a signal received from the modem, or a given time elapse on the built-in real-time clock to trigger an appropriate interrupt routine. Several power statements are also provided, so you can literally tell the computer when to turn itself off.

The Model 100 provides special keys, the graphs and code keys, which allow a total of 255 alphanumeric and graphics characters, including racing cars, a telephone, and other symbols, to be displayed. These can also be produced under program control using CHR$(n) statements.

It's difficult to attempt to go into all the Model 100's features, let alone explore possible uses for the machine. For business executives, it seems ideal. Writers have also latched onto the unit in droves. If nothing else, it makes a handle device to carry text files from a computer at home to a computer at the office.

Some observers feel that the computer is overpriced. At about $1000 for a 32K machine, it is pricey when compared with Commodore 64s selling at $299. The Model 100 is probably not the best value as a first or only computer and should not be compared apples-and-oranges fashion.

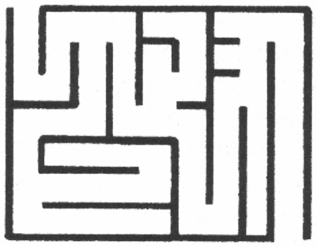
On features alone, the Model 100 is not too far out of line. After all, the memory is not cheap dynamic RAM but CMOS memory that preserves data when the display power switch is off. For your $1000 you get the computer, a modem ($100-$149 more in most computers), RS232 and Centronics interfaces ($49.00-$100), word processing and communications software ($100-$200), plus one of the best BASICs you can buy.

When you include the features that are not available at any price, such as consummate portability, the Model 100 represents a bargain to those who need this combination of advantages. I bought mine the week it was introduced, and I know several other writers who did the same.

With this kind of interest, the Model 100 should not be alone on the market in its price class for long. Displays of 80 characters by eight lines (or even a full 80 × 24) are not far away. Look for more software for the Model 100, video interfaces, and even a tiny disk drive of some sort. The field may get crowded, but this computer will be remembered as a landmark.

As a games player, I appreciate that the machine gives me an excuse to take a personal computer everywhere on business—but I can still have access to it for games.

Chapter 2



Invisible Maze

We all know that the Model 100 is an excellent business machine, and that they are being purchased by hardworking executives who write memos, schedule appointments, or communicate orders with the home office. However, those of us who are deeply immersed in the business computer field also know what those executives do in their spare time: they play games. On one of my first visits to a regional sales office of a major mainframe computer manufacturer in 1974, I found the sales manager hunched over a terminal of a $100,000 computer—playing Othello. And, we all know that computer scientists in the 1960's (and beyond) did with their state-of-the-art equipment. They played Space War, of course.

So, even though the Model 100 is being touted as a "micro executive work station," it is not all work. It can be a pretty fair games machine, and is not limited to word games with no graphics. Invisible Maze is a rudimentary action-oriented race against time to complete a maze. The interesting part is that the maze itself cannot be seen. The player must use the arrow keys to maneuver blindly, progressing from the left side of the liquid crystal screen to the right in the minimum time possible.

The easily-frustrated can breathe comfortably. By pressing the D key, the maze will be displayed briefly. Those with near-photographic memories can improve their recollections of the display by placing the Model 100 near a photograph. The truly chicken can cancel the quest by typing Q. In order to satisfy the desires of everyone, I have even included a sneaky way to cheat. Those who cannot resist the urge to do something immoral can make their way to the "You win!!" prompt without wending through the maze. I won't be telling you what that is, however.

Those who insist on learning something can also benefit from this program. I firmly believe that the best way of learning any BASIC inside out is by typing in programs and then spending a few weeks trying to get them to work. While Invisible Maze works perfectly well in my Model 100, experience has shown me that microcomputer buffs cannot resist the urge to make a few changes as they type.

Some like to change the name of a variable—in most of the places where it occurs. Others like to generate errors by confusing numbers and letters. After all, who in his or her right mind would think that P=l is correct, when P=l (lowercase L) looks almost the same. For you who are more meticulous scholars, the explanation of the program that follows should satisfy your educational needs.

Like most graphics games, Invisible Maze has a requirement for detecting a collision between the cursor and one of the barriers set on the screen. In other computer systems, this is most commonly done by peeking at video memory locations to see if something is stored at the position the cursor will move to next. If so, collision is deemed to have taken place.

The Model 100, on the other hand, doesn't have a video screen. I'll leave whether or not you can peek to see what is on the screen to a later chapter. (The impatient can try peeking from −512 to −192 to see what happens.) It would not be advantageous to detect collisions in this manner anyway, because you hope to keep the maze invisible. Instead, you must set up an array that simulates video memory, with 320 elements to account for each of the 320 print@ locations on the screen.

This array, MZ(n), stores a 0 in any screen position that is empty and a value of 1 in any position that is occupied by a maze block. As the cursor moves throughout the maze, it is simply a matter of checking MZ(n) to see if the next position to be moved to is a 1 or a 0. If it is a 1, the movement is blocked.

That strategy makes it easy to temporarily print out the maze as well. A for-next loop from 1 to 320 prints a graphics block CHR$(239), at any position that shows a 1 in MZ(n).

Let's take a closer look at the program. The

A$	Used in INKEY$ loop
A	Value of character in A$
B1	Position of cursor
DM	Difference in minutes
DS	Difference in seconds
DU	Dummy variable for RND start point
FH	Finish hour
FM	Finish minutes
FS	Finish seconds
MZ(n)	Maze array, keeps track of obstacles
N	Loop counter
NB	Number of obstacles to be set
P	Position of obstacles
Q	Loop counter for sound
SH	Start hour
SM	Start minutes
SS	Start seconds
X	CHR$ code of obstacle
Z	Random sound

Fig. 2-1. Variables used in Invisible Maze.

instructions are printed between lines 80-220, and the player enters the number of walls desired, beginning at line 230. A number between 1 and 9 is entered and is multiplied by 10 to produce between 10 and 90 walls in the maze. If there are more than 90 walls, the odds are that the maze will be unsolvable.

Since the Model 100 generates only pseudorandom numbers, using the same series each time, games that must be different each time require setting a different random number starting point in that series. The most common way of insuring a random starting point is to take the current number of seconds from TIME$ and use that. A for-next loop from 1 to the number of seconds feeds dummy random numbers to DU, just to use them up and allow the program to start from the position following the last dummy random number. Thus, while the number series is the same, sometimes you will start from position number 1; other times, from position number 59 or somewhere in between.

Beginning at line 330, the maze is set up. A for-next loop from 1 to NB, the number of blocks selected chooses random numbers. If a number has already been chosen (line 350), another number is selected. The MZ(n) element is set to a value of 1 for each random number (n). So, if 50 blocks are set, there will be 50 ones scattered throughout MZ(n), while the other 270 elements remain zeros.

The starting time, P$, is taken in line 390, to be used as a measurement of elapsed time to complete the maze. Then, an INKEY$ loop repeats in line 410 until the player pushes a D, to display the maze, Q to quit, or one of the arrow keys, CHR$(28) through CHR$(31). Subroutines at lines 500,540,580, and 620 check to see if the cursor is being moved off the screen, erase the old cursor, and return control to the main routine. There, the MOD function is used to determine whether or not the cursor position, B1, is evenly divisible by 40. If it is, the right side of the screen has been reached, and an appropriate message, along with elapsed time, is displayed.

That's about it. The variables used in the program are shown in Fig. 2-1. And no, I am not going to tell you how to cheat. That would be cheating.

Program Listing

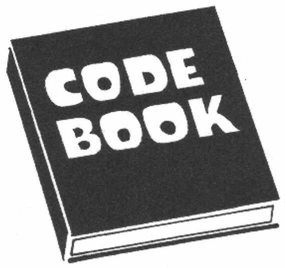
```
 10 '  *******************
 20 '  *                 *
 30 '  * Invisible Maze  *
 40 '  *                 *
 50 '  *******************
 60 '
 70 DIM MZ(320)
 80 X=239

 85 ' *** Instructions ***

 90 CLS:PRINT
100 PRINTTAB(2)"Invisible Maze"
110 PRINT:PRINTTAB(2)"Do you want instructions?"
120 PRINT:PRINTTAB(8)"(Y/N)
```

```
130 A$=INKEY$:IF A$=""GOTO 130
140 IF A$="Y" OR A$="y" GOTO 160
150 GOTO 230
160 CLS:PRINT:PRINT
170 PRINTTAB(8)"== Invisible Maze =="
180 PRINT:PRINTTAB(2)"Use arrow keys to move cursor to"
190 PRINTTAB(2)"right side of screen.  To peek at"
200 PRINTTAB(2)"Maze, hit  ";CHR$(34);"D";CHR$(34);"  key.
    Hit ";CHR$(34);"Q";CHR$(34)" to quit."
210 PRINT:PRINTTAB(6)"== Hit any key to play =="
220 A$=INKEY$:IF A$=""GOTO 220

225 ' *** Enter number of "walls" desired ***

230 CLS:PRINT:PRINT
240 PRINTTAB(2)"Enter difficulty level:":PRINT
250 PRINTTAB(2)"[1] (Easy) though [9] (Hard)"
260 A$=INKEY$:IF A$="" GOTO 260
270 NB=VAL(A$)*10
280 CLS:PRINT@90,"Preparing maze."
290 PRINT@130,"Hold on."

295 ' *** Set random seed ***

300 FOR N=1 TO VAL(RIGHT$(TIME$,2))
310 DU=RND(1)
320 NEXT N

325 ' *** Set up Maze ***

330 FOR N=1 TO NB
340 P=INT(RND(1)*320)
350 IF MZ(P)=1 GOTO 340
360 MZ(P)=1
370 NEXT N
380 CLS
390 B1=1:P$=TIME$
400 PRINT@B1,"*";

405 ' *** Check for keyboard input ***
410 A$=INKEY$:IF A$=""GOTO 410
```

```
420 IF A$="D" OR A$="d" GOTO 660
430 IF A$="Q" OR A$="q" THEN END
440 A=ASC(A$)
450 IF A<28 OR A>31 GOTO 410
460 ON A-27 GOTO 500,540,580,620

465 ' *** If Right Side of Screen ***
      *** Reached, You won.       ***

470 IF B1 MOD 40=0 THEN GOTO 730
480 PRINT @B1,"*";
490 GOSUB 870:GOTO 410

495 ' *** Cursor Right ***

500 IF MZ(B1+1)=1 THEN GOSUB 890:GOTO 410
510 B1=B1+1:IF B1+1>320 THEN B1=320
520 PRINT@B1-1," ";
530 GOTO 470

535 ' *** Cursor Left ***

540 IF MZ(B1-1)=1 THEN GOSUB 890:GOTO 410
550 B1=B1-1:IF B1<1 THEN B1=1
560 PRINT@B1+1," ";
570 GOTO 470

575 ' *** Cursor Up ***

580 B1=B1-40:IF B1<1 THEN B1=B1+40:GOTO 470
590 IF MZ(B1)=1 THEN B1=B1+40:GOSUB 890:GOTO 410
600 PRINT@B1+40," ";
610 GOTO 470

615 ' *** Cursor Down ***

620 B1=B1+40:IF B1>320 THEN B1=B1-40:GOTO 470
630 IF MZ(B1)=1 THEN B1=B1-40:GOSUB 890:GOTO 410
640 PRINT @B1-40," ";
650 GOTO 470
```

```
655 ' *** Display Maze ***

660 FOR N=1 TO 317
670 IF MZ(N)=1 THEN PRINT@N,CHR$(X);
680 NEXT N
690 A$=INKEY$:IF A$=""GOTO 690
700 CLS
710 PRINT@B1,"*";
720 GOTO 410

725 ' *** Game Won ***

730 PRINT@131,"== You win!!! =="
740 SM=VAL(MID$(P$,4,2))
750 SS=VAL(RIGHT$(P$,2))
760 SH=VAL(LEFT$(P$,2))
770 FH=VAL(LEFT$(TIME$,2))
780 FM=VAL(MID$(TIME$,4,2))
790 FS=VAL(RIGHT$(TIME$,2))
800 S1=SS+SM*60+SH*3600
810 S2=FS+FM*60+FH*3600
820 DF=S2-S1:DM=INT(DF/60):DS=DF-DM*60
830 PRINT:PRINTTAB(2)"IT TOOK YOU ";DM;" MIN. AND ";DS;
840 PRINT " SECONDS TO FINISH."
850 INPUT A$
860 RUN

865 ' *** Sound Routine ***

870 SOUND 12000,2
880 RETURN

885 ' *** Collision Sound ***

890 FOR Q=1 TO 10
900 Z=RND(1)*14000
910 SOUND Z,2
920 NEXT Q
930 RETURN
```

Chapter 3



Codebreaker

Games of logic and deduction are popular for personal computers, because they are both easy to program and fun to play. The programmer's chore is easy because no strategy is required on the computer's part. With a game like Codebreaker, it's not necessary to get involved with either heuristics or complex algorithms that calculate the computer's best move.

Instead, the computer can select a code at random and challenge the player to discover what it is. Played something like Mastermind and similar games, Codebreaker offers the player eight chances to deduce the selected four-letter combination.

The code uses only the letters A,B,C,D, and E, but the computer can place the letters in any order, repeat letters, and leave out letters. AAAA, CDDE, and EBCB are all valid codes.

After each guess, the computer provides a clue as to the correctness of the player's entry. A graphic character C indicates that one of the letters entered is the correct letter, in the right position. A P indicates that one letter is right but is located in the wrong position. More than one C or P may be used in the clue. If none of the letters the player has entered are in the code, four dashes, ----, appear.

To help you get the idea, here are some typical entries and the clues that would appear on the Model 100 screen:

Code Selected:	AABB	
First Guess:	BCAD	PP
Second Guess:	AECB	CC
Third Guess:	ABCB	CCP
Fourth Guess:	ACBB	CCC
Fifth Guess:	AABB	CCCC

The secret code is selected in a routine at lines 260-300. The variable C is assigned a value between one and five, which is added to 64 to produce characters in the range CHR$(65)-CHR$(69), which are A to E.

Then, the player is invited to guess the code, which is input through an INKEY$ keyboard strobing loop that accepts a single character in line 330. The entry is then checked to see if the character pressed is CHR$(65) to CHR$(69) or CHR$(97) to CHR$(101). The latter range represents the lowercase letters a to e. Thus, the player does not have to worry about pressing the shift key. The program will accept upper and lowercase letters.

If lowercase letters are input, they are converted to uppercase in line 370, and the character is added to the guess, G$, in line 380. When the length of G$ equals four characters, the program branches to line 420. There, a loop from one to four repeats to see if any character in the guess is identical with any character in the same position in the secret code.

If any one is identical, line 440 adds CHR$ (CO), the graphics C, to the clue string, R$, and the matching letters in both the guess and the code are changed to keep them from triggering the next module, which looks for a correct letter in any position.

In that routine, another for-next loop from one to four uses MID$ to look at each letter of the secret code in turn, storing its value in P$. In line 480, the variable P$ is checked using INSTR to see if P$ appears anywhere in the guess, G$. If it does, then CHR$(PO), the P, is added to the clue, R$.

If no matches are found and R$ is empty, R$ becomes ----, indicating that no letters in the guess appear in the code. A check is then made to see if R$ happens to equal CCCC; in which case, the code has been deciphered and the program branches to line 610. There, the number of turns the player used to decipher the code is displayed.

Otherwise, the program loops back for another guess, until eight guesses have been entered. Then, the player is informed of the bad news and invited to play again.

There is one Model 100 programming trick used in this program that you might have missed. Lines 70 and 80 look a bit odd, don't they?

```
70 PRINTTAB(12)CHR$(27);"p";
   "Codebreaker"

80 PRINTCHR$(27)"q" :PRINTTAB
   (12)"Instructions?"
```

A	ASCII value of the player's guess, used to convert letters to uppercase if necessary
A$	Used in INKEY$ loop
C	Random number used to select the secret code
C$	Code selected by the computer
CO	ASCII code used to print the C in the clue string
DU	Dummy variable for RND start point
G$	Player's guess
N	Loop counter
P$	Used to determine if any correct letters are included in the player's guess
PO	ASCII code used to print the P in the clue string
R$	Clue string
T$	Code string used in checking player's guess
Turn	Number of guesses taken

Fig. 3-1. Variables used in Codebreaker.

What these two lines do is invoke escape codes within the program. The escape key is that key in the upper left-hand corner of the Model 100 keyboard. Many programs, time-sharing services, and other computer setups look for this key, which is CHR$(27). Anything following that key is interpreted as an *escape sequence*. The Model 100 recognizes several escape codes. ESCAPE-p (it must be a lowercase p; an uppercase P will not work) turns on the reverse character mode. ESCAPE-q (again, an upper Q will not work) turns reverse mode off.

I have heard these modes referred to as *reverse video*, because most computers using them do implement them by reversing the video output. However, the Model 100 doesn't have video at all—or have you forgotten? Somehow, *reverse LCD* does not sound the same.

To add reverse mode to your own programs, simply enter CHR$(27) at the proper point, and the lowercase p. Note that all of the screen following the code will be reversed; even the spaces will appear black. So, if you want only the letters to be reversed on a light-colored screen, it will be necessary to repeatedly enter ESC-p and ESC-q at the proper points.

Also note that the semicolon following the CHR$(27) is optional. It has been included on line 70, but deleted from line 80; both work.

A listing of the variables used in the program is shown in Fig. 3-1.

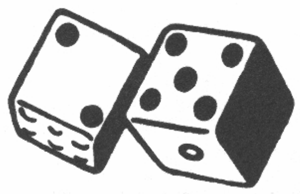
Program Listing

```
 10 ' ***************
 20 ' *             *
 30 ' * Codebreaker *
 40 ' *             *
 50 ' ***************

 55 ' *** Instructions ***

 60 CLS:PRINT
 70 PRINTTAB(12)CHR$(27);"p";"Codebreaker"
 80 PRINTCHR$(27)"q":PRINTTAB(12)"Instructions?"
 90 PRINT:PRINTTAB(16)"Y/N"
100 A$=INKEY$:IF A$=""GOTO 100
110 IF A$="Y" OR A$="y" GOTO 120 ELSE GOTO 200
120 CLS:PRINT
130 PRINT:PRINTTAB(2)"Try to guess the computer's 4-letter"
140 PRINTTAB(2)"Code in fewer than 8 guesses.  Code"
150 PRINTTAB(2)"uses";CHR$(34);"ABCDE.";CHR$(34);:PRINTTAB
    (2)"A ";CHR$(34);CHR$(171);CHR$(34);" indicates the"
160 PRINTTAB(2)"right letter in right position. A ";
    CHR$(34);"P";CHR$(34)
170 PRINTTAB(2)"indicates right letter,wrong position."
180 PRINTTAB(12)"== Hit any key =="
190 A$=INKEY$:IF A$="" GOTO 190
```

```
195 ' *** Set random start point ***

200 FOR N=1 TO VAL(RIGHT$(TIME$,2))
210 DU=RND(1)
220 NEXT N
230 CO=171:PO=80
240 CLS
250 C$="":G$=""

255 ' ***  Choose Secret  Code ***

260 FOR N=1 TO 4
270 C=INT(RND(1)*5)+1
280 C$=C$+CHR$(C+64)
290 NEXT N
300 T$=C$

305 ' *** Get Player Guess ***

310 PRINTTAB(2)"Guess:"
320 PRINTTAB(8)"";
330 A$=INKEY$:IF A$=""GOTO 330
340 A=ASC(A$)
350 IF A=65 OR A=66 OR A=67 OR A=68 OR A=69 OR A=97 OR A=98
    OR A=99 OR A=100 OR A=101 GOTO 370
360 GOTO 320
370 IF A>96 THEN A=A-32
380 G$=G$+CHR$(A)
390 PRINT CHR$(A);
400 IF LEN(G$)=4 GOTO 420
410 GOTO 320

415 ' *** Check for Right Letter, Position ***

420 PRINTTAB(16)"";
430 FOR N=1 TO 4
440 IF MID$(G$,N,1)=MID$(T$,N,1)THEN R$=R$+CHR$(CO)
    :MID$(G$,N,1)="0":MID$(T$,N,1)="1"
450 NEXT N

455 ' *** Check for Right Letter, Wrong Position ***
```

```
460 FOR N=1 TO 4
470 P$=MID$(T$,N,1)
480 I=INSTR(G$,P$):IF I=0 GOTO 510
490 R$=R$+CHR$(PO)
500 MID$(G$,I,1)="0"
510 NEXT N
520 TURN=TURN+1
530 IF R$="" THEN R$="----"
540 PRINT R$
550 IF R$=STRING$(4,CO) GOTO 610
560 IF TURN=8 GOTO 630
570 G$=""
580 R$=""
590 T$=C$
600 GOTO 320

605  ' *** Code Guessed ***

610 PRINT:PRINTTAB(2)"Correct, in ";TURN;" guesses."
620 PRINT:GOTO 660

625 ' *** Code Not Guessed ***

630 PRINT
640 PRINTTAB(2)"Sorry, but you had your eight guesses!"
650 PRINT:PRINTTAB(2)"Correct code was:";C$
660 PRINT:PRINTTAB(2)"Play again?"
670 PRINTTAB(10)"(Y/N)"
680 A$=INKEY$:IF A$=""GOTO 680
690 IF A$="Y" OR A$="y" THEN RUN
```

Chapter 4



Avarice

Dice games have always been popular for personal computers, because the laws of probability are easy to understand, and pseudorandomness is built into every micro in some form or another. Avarice is an adaptation of an old classic and is written for two players.

The object is to start with $200 and accumulate $500 before your opponent does. At the start of each round, the dice are rolled once, and that roll becomes the player's point for that round. Unlike craps, however, the object is to get as many points as possible before the point comes up again. The player may elect to stop rolling at any point and collect his or her winnings. Or the participant may choose to continue, risking all. If the point is rolled again, the pot is subtracted from the player's total.

An easy point, such as 6, 7, or 8, may crap out early, after only one or two rolls. Getting a 12 or a 2 as a point can tempt the player to roll again and again, amassing several hundred dollars on one turn. As stakes get higher, risks increase as well.

This game takes advantage of several of the Model 100's print features to make a neat display. The print using statement formats the dollar and cents output with proper decimal points, while print @ is used to place only the new values on the screen, while avoiding an annoying rewriting of the entire screen between rolls.

The game commences with the obligatory presentation of instructions, which most never read anyway. You can delete them from the game to save space if you wish.

Next, a random starting place in the Model 100's pseudorandom number series is chosen. This is accomplished between lines 250-270 by choosing dummy random numbers, the number of which depends on the number of seconds on the real time clock when the game is begun. That way, RND will sometimes start at the second position in its series and sometimes at the 60th.

The dice rolling subroutine, lines 290-320, forms the heart of the game. Two die, D1 and D2,

A$	Used in INKEY$ loop
AN$	Player answer
CASH(n)	Amount of player's money
D1	Die #1
D2	Die #2
DU	Dummy variable for RND(1)
F$	Print using format
FR	First roll
N	Loop counter
OP	Opponent
PLAYER	Current player
ROLL	Most recent roll of dice
TT	Total of rolls this round

Fig. 4-1. Variables used in Avarice.

are produced by choosing random numbers between 1 and 6, and adding them together to come up with the value for the variable ROLL.

At the start of the round, the first ROLL becomes FR (line 340). On subsequent rolls, if ROLL=FR, then the round is lost, and the program branches to the "you lost!" routine at line 670. Otherwise, the number of points rolled is added to the pot. You can see that really large additions to the pot, such as 11 or 12, will be somewhat rare. But, so will small ones, such as 2 or 3. Most rolls will be in the 6-8 range.

If a player chickens out, the current pot total, TT, is added to the player's cash, and TT is zeroed. If the point comes up, the amount in TT is subtracted from the player's total. At appropriate points, player totals are compared to see if either has over $500.00. If so, the "you win" routine is activated. Figure 4-1 shows the variables used in the program.

You can modify this game to raise the total needed to win or to lower the amount the players start with. Because it is pretty difficult to lose the initial $200, I did not write a line to check for negative totals. Obviously, a player should lose in this situation. As written, however, it is possible to make a comeback from a negative score and even win. If you like really long games, you might want to raise the winning total required to $1000 or more. I have found that it is possible for a player to win on his or her first turn. Lucking out with a 2 or 12 as a point can mean piling up the necessary $300 with a fortuitous string of rolls. Setting the winning total at $1000 would eliminate this possibility. I elected to leave the total lower because of the attraction of the occasional "go for broke" risk.

Program Listing

```
10 ' *************
20 ' *           *
30 ' * Avarice    *
40 ' *           *
50 ' *************
```

```
 55 ' *** Instructions ***

 60 CLS:PRINT:PRINT
 70 PRINTTAB(6)"Do you want instructions?"
 80 PRINT:PRINTTAB(14)"(Y/N)"
 90 A$=INKEY$:IF A$=""GOTO90
100 IF A$="Y" OR A$="y" GOTO120
110 GOTO200
120 CLS:PRINT:PRINT
130 PRINTTAB(1)"Two players try to accumulate 500"
140  PRINTTAB(1)"points to win. Each turn, they roll"
150 PRINTTAB(1)"dice, gaining points until they choose"
160 PRINTTAB(1)"to collect their pot,or else they roll"
170 PRINTTAB(1)"their point again.  If so,the pot is"
180 PRINTTAB(1)"subtracted from their score."
190 A$=INKEY$:IFA$="" GOTO190
200 CLS
210 PLAYER=1
220 OP=2
230 CASH(1)=200:CASH(2)=200
240 F$="$$####.##"

245 ' *** Set Random Starting Point ***

250 FORN=1 TO VAL(RIGHT$(TIME$,2))
260 DU=RND(1)
270 NEXT N
280 GOTO330

285 ' *** Roll Dice ***

290 D1=INT(RND(1)*6)+1
300 D2=INT(RND(1)*6)+1
310 ROLL=D1+D2
320 RETURN

325 ' *** Start Round ***

330 GOSUB290
340 FR=ROLL
350 GOSUB290
```

```
360 IF FR=ROLL GOTO350
370 GOTO400
380 GOSUB290
390 IF FR=ROLL GOTO580
400 PRINT @22,"     ":PRINT@60,"     "
410 IF CASH(1)>500 OR CASH(2)>500 GOTO670
420 PRINT@10,"First roll: ";FR
430 PRINT @50,"Next roll:  ";ROLL
440 TT=TT+ROLL
450 PRINT @122,"Your Total:";:PRINTUSINGF$;CASH(PLAYER)
460 PRINT@165,"Pot:";:PRINTUSINGF$;TT
470 PRINT @187,"Player #";PLAYER
480 PRINT @145,"Opp:";:PRINT USING F$;CASH(OP)
490 PRINT@245,"Roll again?"
500 AN$=INKEY$:IF AN$=""GOTO500

505 ' *** Pot Added to Player ***

510 IF AN$="N" OR AN$="n" THEN
    CASH(PLAYER)=CASH(PLAYER)+TT:GOTO540

515 ' *** Otherwise, Roll Again ***

520 IF AN$="Y" OR AN$="y"THENPRINT@245,SPACE$(12);:FOR N=1 TO
10:NEXT N:GOTO380
530 GOTO500

535 ' *** Change Players ***

540 TT=0
550 IF PLAYER=1 THEN PLAYER=2:OP=1:GOTO330
560 OP=2
570 PLAYER=1:GOTO330

575 ' *** Pot Subtracted from Player ***

580 CLS:PRINT:PRINT
590 PRINTTAB(15)"You lose!"
600 PRINTTAB(12)"Total now:";
610 CASH(PLAYER)=CASH(PLAYER)-TT:PRINTUSING F$;CASH(PLAYER)
620 PRINT:PRINTTAB(13)"== Hit any key =="
```

```
630 A$=INKEY$:IF A$=""GOTO630
640 TT=0
650 CLS
660 GOTO540

665 ' *** Game Over, Display Results  ***

670 CLS:PRINT:PRINT
680 IF CASH(1)>CASH(2)THENPRINTTAB(8)"Player 1 wins" ELSE
    PRINTTAB(8)"Player 2 wins"
690 PRINTTAB(8)"with ";
700 IF CASH(1)>500 THEN PRINT USING F$;CASH(1)ELSE PRINT
    USINGF$;CASH(2);:PRINT"."
710 PRINT:PRINT
720 PRINTTAB(6)"Would you like to play again?"
730 PRINT:PRINTTAB(12)"(Y/N)"
740 A$=INKEY$:IF A$=""GOTO740
750 IF A$="Y" OR A$="y" THEN RUN
```

Chapter 5

Road Rally

As noted in Chapter 2, nearly all graphics games programs require some way of detecting a collision between the player's ship or missile and the enemy ships or missiles. Some more sophisticated BASICs have collision detection between sprites as a regular function. With other BASICs, you need to peek to see if something is displayed on the screen in the location where your object is planning to move next.

Last time, in the discussion of Invisible Maze, I hinted at the possibility of peeking in some location in the Model 100's memory to see what is shown on the screen. Road Rally takes advantage of that capability to see whether the racing car is on the road or has taken an ignominious detour onto the berm, shoulder, or what have you.

The model 100 stores a record of what is displayed on the LCD screen in memory locations 32959 to 33279, in reverse order. That is, 32959 keeps track of what is in the 320th print @ position on the screen (row 8, column 40), while 33279 points to the value at print @ position 0 (row 1, column 1). With a reverse for-next loop, that is, FOR N=32959 to 33279 STEP −1, you could peek at the memory locations in proper order, except for one thing: with many computers, peek only accepts decimal integers and, as you know, these are limited to numbers no higher than 32767.

To get around this limitation, most TRS-80's use negative numbers to peek memory locations higher than 32767. That is, to see 32768, you type PEEK(−1). Then, PEEK(−2) will show you 32769, and so forth up to −32767, which is the very top of available Model 100 memory addresses. However the Model 100 can use either negative numbers or numbers up to 65535 when peeking. To retain compatability with other TRS-80 computers, the negative number system is used in this book.

In this form, the memory locations you are interested in are −512 to −192, and a for-next loop will peek those values with no problem. Try this program to see what happens:

```
 10 DIM SC(320)
 20 FOR N=1 to 8
 30 PRINT"Filling screen with
    alpha characters."
 40 NEXT N
 50 FOR N=512 TO 192 STEP -1
 60 CU=CU+1
 70 SC(CU)=PEEK(-N)
 80 NEXT N
 90 CLS
100 FOR N=1 TO 320
110 PRINT CHR$(SC(N));
120 NEXT N
```

With Road Rally, it is not necessary to peek the whole screen. You only need to look at the position the racing car will move to next, which is limited to the top row of the screen, as shown in Fig. 5-1. So, two variables keep track of the print @ location you use to print the car (P) and the memory location to be peeked (S). As the car moves right, P and S are incremented. As the car moves left, P and S are decremented by one. Before each movement, you can peek to see if S contains a space (32) or one of the graphics blocks that make up the side of the road (CHR$(239)). If a space is not found, the car has crashed.

The program starts off by presenting instructions and asking the player to enter a difficulty level between 1 and 9. This factor, A, is added to 3 to establish the width of the road (W), which can be 3 to 12 spaces wide (some players may enter 0 or hit a key other than a number).

Then, the current time is stored in TM$ (line 330), and the car character, C$, is defined as CHR$(132). This is the handy racing car built into the Model 100 character set. P$ is defined as CHR$(239), which is the solid graphics block. The initial position of the car is defined as print @ location 56, and S is set to correspond. The left edge of the road is positioned at print @ position 295, and the right edge is located W positions to the right.

Play begins when an INKEY$ loop looks for keyboard input, beginning at line 410. Each time through this loop, several things happen. The car is

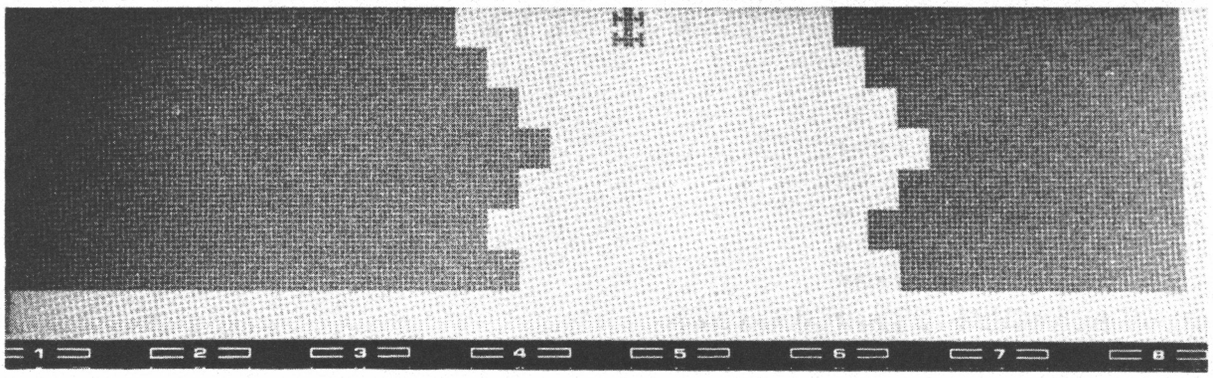


Fig. 5-1. Screen display from Road Rally.

A$	Used in INKEY$ loop
A	Value of A$
C$	CHR$ of car graphics character
DF	Elapsed time difference
EM	Elapsed minutes
ES	Elapsed seconds
F	Finish time
FH	Finish hours
FLAG$	Checkered flag
FM	Finish minutes
FS	Finish seconds
L	Left position limit
N	Loop counter
P$	Pylon graphics character
P	Position of car
R	Right position limit
SH	Start hours
SM	Start minutes
SS	Start seconds
TM$	Starting TIME$
W	Width of road

Fig. 5-2. Variables used in Road Rally.

printed @ P, and a STRING$ of solid blocks are printed from the lower left of the screen up to L, the beginning of the road. More blocks are placed after R, the right side, and extend to the right edge of the screen. As the blocks on either side of the road scroll up, they present a pathway for the car.

No race would be a challenge if the road remained straight. So, the road moves one block to the right or left each time through. The direction is determined by a random number chosen in line 460. In lines 490-500, a check is made to make sure the roadway is not scrolling off either side of the screen.

Finally, the program peeks at S to see if the car has driven into a graphics block. If it has, control goes to the "You crashed!" routine. Otherwise, if no key has been pressed, the program makes a detour to line 810 for a bit of sound and then loops back to line 410 to repeat the process.

When either the right or left arrow keys (CHR$(28) or CHR$(29)) is pressed, control branches to either 560 or 590, where both B and S are adjusted appropriately.

That's about all there is to the game. A list of variables is found in Fig. 5-2. After the crash and some random sounds, the finishing time, converted to seconds, is compared to the starting time, which is also in seconds. The elapsed time is converted back to minutes and seconds, and the player is told how long he or she was able to drive before totaling the car.

I thought play was slow enough in this game that a delay loop was not necessary, but you can add one if you wish. You might also want to write a routine to produce other cars on the road, at random positions larger than L but smaller than R, and then add a check to see if anything other than 32 is stored in S. If so, crash!

Program Listing

```
10 ' **************
20 ' *            *
30 ' * Road Rally *
40 ' *            *
50 ' **************
60 FLAG$=CHR$(239)+CHR$(32)
70 CLS
80 GOTO130
```

```
 85 ' *** Logo and Instructions ***

 90 FORN=1 TO 20:PRINT FLAG$;:NEXT N
100 PRINT CHR$(32);
110 FOR N=1 TO 19:PRINT FLAG$;:NEXT N
120 RETURN
130 GOSUB90
140 PRINT:PRINT:PRINTTAB(8)" == Road Rally =="
150 PRINT:PRINTTAB(9)"Instructions? (Y/N)"
160 GOSUB90
170 A$=INKEY$:IF A$=""GOTO170
180 IF A$="Y" OR A$="y" GOTO200
190 GOTO260
200 CLS:PRINT
210 PRINTTAB(4)"Use arrow keys to stay on road."
220 PRINTTAB(4)"Try to beat your best time."
230 PRINT:PRINT
240 PRINTTAB(6)"== Hit any key to play =="
250 A$=INKEY$:IF A$="" GOTO250
260 CLS:PRINT:PRINT
270 PRINTTAB(8)"Enter difficulty level:"
280 PRINT
290 PRINTTAB(8)"[1] (Hard) to [9] (Easy)"
300 A$=INKEY$:IF A$=""GOTO300
310 A=VAL(A$)
320 W=3+A
330 TM$=TIME$
340 CLS
350 C$=CHR$(132)
360 P$=CHR$(239)
370 P=56
380 S=-512+P
390 L=295
400 R=L+W

405 ' *** Keyboard Loop ***

410 A$=INKEY$
420   PRINT@P,C$
430   PRINT@280,STRING$(L-280,P$);
440   PRINT@R,STRING$(318-R,P$);
```

```
450   PRINT
460   N=INT(RND(1)*2)
470   IF N=0 THEN N=-1
480   L=L+N:R=R+N
490   IF R>318 THEN R=318:L=R-W:GOTO510
500   IF L<280 THEN L=280:R=L+W:GOTO510
510   IF PEEK(S)=239 GOTO620
520   GOSUB810
530 IF A$=""GOTO410
540 A=ASC(A$):IF A<28 OR A>29 GOTO 410
550 ON A-27 GOTO560,590

555 ' *** Move Car Right ***

560 P=P+1
570 S=S+1
580 GOTO410

585 ' *** Move Car Left ***

590 P=P-1
600 S=S-1
610 GOTO410

615 ' *** Crashed -- Game Over ***

620 CLS:PRINT
630 FOR N=1 TO 50:SOUND(RND(1)*10000),1:NEXT N
640 PRINTTAB(8)"You crashed!":PRINT

645 ' *** Calculate Elapsed Time ***

650 SH=VAL(LEFT$(TM$,2))
660 SM=VAL(MID$(TM$,4,2))
670 SS=VAL(RIGHT$(TM$,2))
680 FH=VAL(LEFT$(TIME$,2))
690 FM=VAL(MID$(TIME$,4,2))
700 FS=VAL(RIGHT$(TIME$,2))
710 S=SH*3600+SM*60+SS
720 F=FH*3600+FM*60+FS
730 DF=F-S:IF DF<60 THEN EM=0:ES=DF:GOTO760
```

```
740 EM=INT(DF/60)
750 ES=DF-EM*60
760 PRINTTAB(2)"You lasted ";EM;" minutes";ES;" seconds."
770 PRINT:PRINTTAB(8)"Play again?"
780 PRINT:PRINTTAB(8)"(Y/N)";
790 INPUT A$
800 IF A$="Y" OR A$="y" THEN RUN ELSE END
810 SOUND 5000,1
820 RETURN
```

Chapter 6

Computer Magi

Everyone likes parlor tricks, even those under 30 who are not quite sure what a parlor is. The TRS-80 Model 100, however, is just as at home in the parlor as in the office, and this magic trick can be your high-tech entree to computerized legerdemain.

This amazing trick uses misdirection, facile patter, and the magic fingers of a touch typist to produce the illusion that the computer is actually reading the minds of the spectators. Although, like most magic tricks, it is effective when done once or perhaps twice, there are a few features built in that make it suitable for those who are easily carried away when they find themselves at the center of attention.

The illusion is thus: the computer operating wizard, seated at the keyboard (or, standing holding the Model 100 if you will) types in a challenge to the computer to answer a question thrown out by the audience. There, as if by magic, appears the answer! The Magi risks being thrown out by the audience if the secret is unveiled, but here it is nevertheless.

Although the operator appears to be typing in an innocuous message to the computer, he or she is actually typing in the answer at that time. What appears on the screen is not an echo of what is being typed. The innocuous messages are stored in the program itself, and displayed, a character at a time, as each letter of the answer is entered. Clever, no?

What happens is that the computer reads a message into string variable MESSAGE$ in line 110. Then, an INKEY$ loop looks for a key to be pressed. The key that is entered is stored in AN$, while one letter of MESSAGE$ is printed to the screen. Which letter is determined by a counter, CU, that is incremented every time a key is pressed. When the answer is finished, the Magi enters a period (the little dot at the end of a sentence). This sets FLAG to one and tells the program to stop adding any additional input to AN$. In that way, the answer to the question can be shorter than the message that is being printed to the screen. A listing of the variables used in the program is provided in Fig. 6-1.

AN$	Answer to audience question
CU	Counter of how many characters printed
FLAG	Flag indicating whether or not period entered
KBD$	Used in INKEY$ loop
MESSAGE$	Message read from data
N	Loop counter

Fig. 6-1. Variables used in Computer Magi.

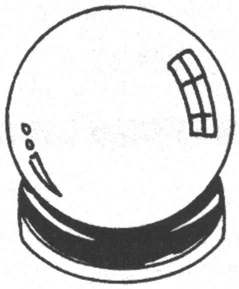
Eight messages are provided. You can personalize the program by substituting your own messages. It is possible to increase the number of messages displayed by changing the eight in line 55 to a larger number. If the operator gets carried away and exceeds the number of messages available, a restore command in line 280 sets the data pointer back to the beginning and the messages begin repeating. By this point, the audience will have caught on, anyway.

Program Listing

```
 10 ' *****************
 20 ' *               *
 30 ' * COMPUTER MAGI *
 40 ' *               *
 50 ' *****************
 55 FOR N=1 TO 8
 60 CLS:PRINT
 65 PRINTTAB(2)"";
110 READ MESSAGE$
120 KBD$=INKEY$
130 IF KBD$="" GOTO 120
140 IF KBD$=CHR$(13) GOTO 200
150 IF KBD$="." THEN FLAG=1
160 IF FLAG<>1 THEN AN$=AN$+KBD$
170 CU=CU+1
180 PRINT MID$(MESSAGE$,CU,1);
190 GOTO 120
200 PRINT:PRINTTAB(2)AN$
205 PRINT:PRINT
210 PRINTTAB(2)"Hit any key for another question."
230 KBD$=INKEY$:IF KBD$="" GOTO 230
240 AN$=""
250 CU=0
260 FLAG=0
270 NEXT N
```

```
280 RESTORE
290 GOTO 55
300 DATA "O great computer, speak!"
310 DATA "Enlighten us, Computer Magi."
320 DATA "See if you can guess this."
330 DATA "Ready for another one?"
340 DATA "Give us the answer, please."
350 DATA "Not bad for a dumb machine."
360 DATA "Right again, amazing one."
370 DATA "Last question. Ready?"
```

Chapter 7



Swami

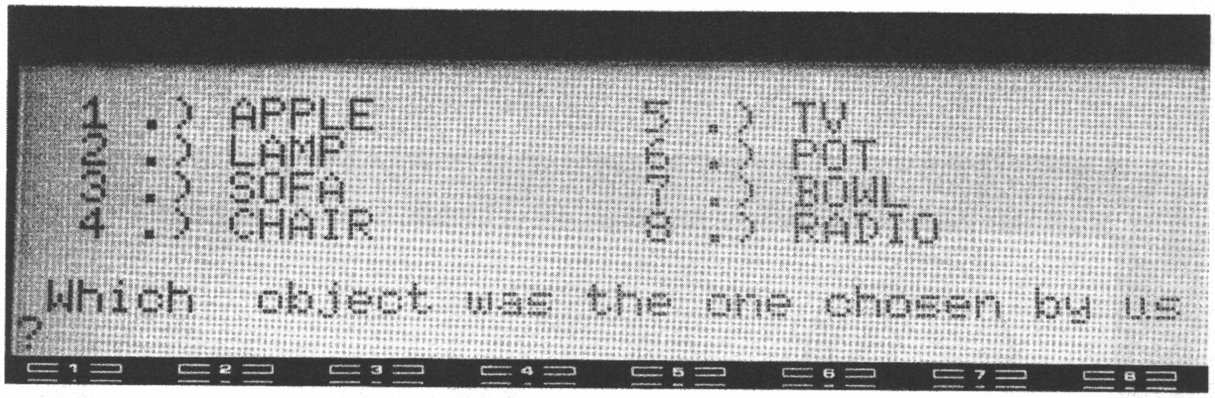
Probably the only question asked of amateur magicians more than "How did you do that?" is "Is that the only trick you can do?" If posed that query after a dazzling demonstration of Computer Magi, simply load Swami and prove that your display was not a fluke. In this game, the magician reads the computer's mind. In most cases, this will be a more substantial task, anyway.

The trick works like this. The magician leaves the room for a short time. Warning: only perform this trick in front of an audience you trust. I personally have had several airline flight attendants attempt to run off with my Model 100. While the Swami is out of earshot and eyeshot, the audience, or preferably one member who can type, will enter the names of eight common objects found in the room, such as PEN, FLOOR, CRAY-2, etc. Given the space constraints of the Model 100 screen, these objects should not have long names. Warn the audience in advance that if some joker suggests wall-to-wall carpeting, you will flay him/her.

The participants will then select one of the objects as the item that the Swami will attempt to pick out from all the others, or the audience can select to have the computer choose an object and tell no one. In that way, the gathering can be assured that the Swami read the computer's mind and did not cheat by reading the mind of someone in the audience instead.

When the Swami returns to the room, he/she (Please note: nonsexist writing is very cumbersome. Or is it Swama?) glances at the computer with a confident look and picks out the object selected—without even touching the computer.

No, this is not a hardware problem. The tipoff for the Swami is the seemingly innocuous message, "Which object was the one chosen by us?" Innocuous is the standard tool of the magician. Each word in the sentence is separated from the next by a space—except for one word, which is followed by two spaces, as shown in Fig. 7-1. Although this extra space sticks out like a sore thumb, most will



```
1 .) APPLE                5 .) TV
2 .) LAMP                 6 .) POT
3 .) SOFA                 7 .) BOWL
4 .) CHAIR                8 .) RADIO
Which  object was the one chosen by us
?
```

Fig. 7-1. Screen display from Swami.

not notice it or connect it with the trick unless the magician is stupid and repeats the stunt a few times.

If object 1 is chosen, the first word in the sentence will have two spaces after it. If object 2 is chosen, the second word will have two spaces. Do I have to spell it out for you?

The program itself is easy to understand, at least for anyone capable of pulling the trick off. A list of the variables used is found in Fig. 7-1. A for-next loop from 1 to 8 in lines 210-270 asks for input of object names. Then the audience is requested to choose an object or ask the computer to do so. The participant's choice is stored in variable NU, or the computer chooses a random number in line 380 and stores this in NU instead.

The eight objects are displayed on the screen in two columns, since the Model 100 has only eight lines. A for-next loop from 1 to 4 is used to print both OBJECT$(N) and OBJECT$(N+4) on a single line. The innocuous message is printed to the screen in a routine beginning at line 540. Here, each word is read from the data lines one at a time, and if the loop counter equals NU an extra space printed.

Then the magician reveals his/her choice, either by telling it out loud or writing it down on a piece of paper in a sealed envelope (a popular device used to raise audience suspicions) (this is called misdirection). The computer then gives the answer. Polite applause or excited cheering ensues, followed by a request for another trick. Since this is the last trick in the book, the magician would be well advised to quit while he/she is ahead.

A$	Used in INKEY$ loop
CH	Player's choice as to who chooses
DU	Dummy variable for RND(1)
N	Loop counter
NU	Number of object selected
OBJECT$(n)	Array storing names of objects
WRD$	Stores words read from data

Fig. 7-2. Variables used in Swami.

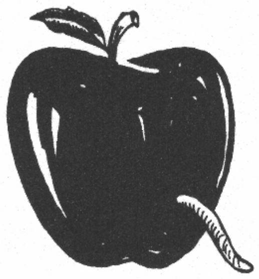
Program Listing

```
 10 ' ****************
 20 ' *              *
 30 ' *    SWAMI     *
 40 ' *              *
 50 ' ****************

 55 ' *** Set random start point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N
 90 CLS:PRINT:PRINT
100 PRINTTAB(8)"Do you want instructions?"
110 PRINT:PRINTTAB(16)"(Y/N)"
120 A$=INKEY$:IF A$=""GOTO 120
130 IF A$="Y" OR A$="y" GOTO 140 ELSE GOTO 190
140 CLS:PRINT

145 ' *** Instructions ***

150 PRINT"WHEN THE SAGE IS OUT OF THE ROOM ENTER"
160 PRINT"NAMES OF 8 COMMON OBJECTS IN THIS ROOM."
170 PRINT"YOU MAY THEN CHOOSE ONE & THE SAGE WILL"
180 PRINT"ATTEMPT TO DETERMINE YOUR CHOICE BY ESP."
190 PRINTTAB(6)"== Hit any key to begin =="
200 A$=INKEY$:IF A$=""GOTO 200

205 ' *** Enter Names of Objects ***

210 CLS:PRINT
220 FOR N=1 TO 8
230 CLS:PRINT:PRINT
240 PRINTTAB(4)"ENTER NAME OF OBJECT #";N;":"
250 PRINTTAB(4)"";
260 INPUT OBJECT$(N)
270 NEXT N

275 ' *** Computer/Players Choose ***

280 CLS:PRINT:PRINT
```

```
290 PRINTTAB(4)"WOULD YOU LIKE:"
300 PRINT
310 PRINTTAB(8)"1.) TO CHOOSE OBJECT"
320 PRINTTAB(8)"2.) COMPUTER CHOOSE"
330 PRINT
340 PRINTTAB(12)"ENTER CHOICE :"
350 A$=INKEY$:IF A$=""GOTO 350
360 CH=VAL(A$)
370 IF CH<1 OR CH>2 GOTO 350

375 ' *** Choose Object ***

380 IF CH=2 THEN NU=INT(RND(1)*8)+1:GOTO480
390 CLS:PRINT:PRINT
400 FOR N=1 TO 4
410 PRINTTAB(1)N;".) ";OBJECT$(N);TAB(20)N+4;".) ";OBJECT$
    (N+4)
420 NEXT N
430 PRINT
440 PRINTTAB(2)"Enter number of object to be found:"
450 A$=INKEY$:IF A$=""GOTO 450
460 NU=VAL(A$)
470 IF NU<1 OR NU>8 GOTO 450

475 ' *** Display on Screen ***

480 CLS:PRINT
490 FOR N=1 TO 4
500 PRINTTAB(1);N;".) ";OBJECT$(N);TAB(20)N+4;".) ";OBJECT$
    (N+4)
510 NEXT N
520 PRINT
530 PRINTTAB(1)"";
540 FOR N=1 TO 9
550 READ WRD$
560 PRINTWRD$;CHR$(32);
570 IF N=NU THEN PRINT CHR$(32);
580 NEXT N
590 A$=INKEY$:IF A$=""GOTO 590
600 IF CH=2 GOTO 650

605 ' *** Give answer ***
```

```
610  CLS:PRINT:PRINT
620  PRINTTAB(2)"Audience chose :";OBJECT$(NU)
625  PRINT:PRINTTAB(12)"Try again?"
626  PRINTTAB(14)"(Y/N)"
630  A$=INKEY$:IF A$=""GOTO 630
635  IF A$="Y" OR A$="y" THEN RUN
640  END
650  CLS:PRINT:PRINT
660  PRINTTAB(4)"COMPUTER CHOSE ";
670  PRINT OBJECT$(NU)
675  PRINT:PRINTTAB(12)"Try again?"
676  PRINTTAB(14)"(Y/N)"
680  A$=INKEY$:IF A$=""GOTO 680
685  IF A$= "Y" OR A$="y" THEN RUN
690  END
700  DATA Which,object,was,the,one,chosen,by,us,?
```

Chapter 8



Worm Hole

Back in the dark ages of microcomputers (1981 and before), most of the games available for home or hobbyist machines were stupid word games. Business computers, the precursors of our micro executive work station, were generally even worse off. Only the computer programmers themselves had access to good Star Trek games.

Although this book has its share of classics, I have tried to include some graphics-laden arcade style games, within the constraints of the Model 100's screen. Invisible Maze and Road Rally were two such. Now meet Worm Hole, another chance to move around on the screen at breakneck speeds in a vain attempt to control the movement of an unruly cursor.

In this game, the object is to avoid obstacles on the screen for as long as possible. This is tricky, because your worm leaves its own trail, to be avoided at all costs. You get one point for every position you move on the screen without hitting anything.

The player gets to determine just how hard the game is. He or she is invited to enter the speed you want in the range from 1 (Fast) to 9 (Slow). This input actually sets a variable, DL, which is used for a delay loop from 1 to DL. If DL is large, the loop will be long. If DL is small, the loop will be short and therefore, quick. Got it?

Difficulty level, also from 1 to 9, can be set. This variable determines the number of obstacles printed to the screen in lines 450 to 470.

Play begins at line 480, where one of the ubiquitous INKEY$ loops looks for keyboard input. Each time through the loop, the position of the cursor, B1, is either incremented or decremented, depending on the value of DELTA. If the last arrow key pressed was the right arrow, then DELTA will equal 1 and B1 will be increased to one position to the right. If the last arrow key pressed was the left arrow, then DELTA will equal −1, and B1 will become one less, moving the cursor to the left. When DELTA is +40, B1 will be one whole line

A$	Used in INKEY$ loop
A	Value of A$
B1	Position of cursor
C$	Cursor character
D	Difficulty, number of obstacles
DELTA	Direction of cursor travel
DL	Delay loop variable
DU	Dummy variable for RND(1)
N	Loop counter
N1	Loop counter
PT	Points accumulated
T$	Instructions prompt

Fig. 8-1. Variables used in Worm Hole.

larger, and movement will be downward. If DELTA equals −40, the movement will be upward.

Appropriate checks are made to see that B1 is not being changed to a number larger than 320 or smaller than 1, both beyond the range of the print @ commands which send the cursor scuttling around the screen. As long as no arrow key is pressed, DELTA remains the same, and the cursor continues to move in the same direction. It is not necessary to hit a key for each movement.

Speed of movement is controlled by the delay loop in line 550, and each movement increments PT (points) by one, increasing the player's score.

In line 520, a collision is detected by checking to see if the cursor position is occupied, using the peek technique discussed earlier.

Line 540 makes things tolerable for the player by printing random holes in the screen to open new escape routes. The arrow keys are checked for in line 590. If one has been pressed, control is sent to one of four routines located between lines 610-640, which alter the value of DELTA and send the cursor in the new direction. A list of variables used in the program is found in Fig. 8-1.

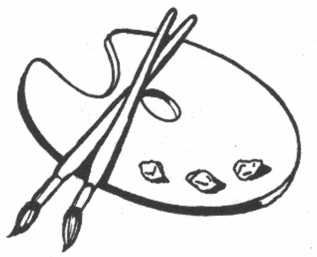
Program Listing

```
 10 ' *************
 20 ' *           *
 30 ' * Worm Hole *
 40 ' *           *
 50 ' *************

 55 ' *** Instructions ***

 60 T$="Instructions?"
 70 CLS:PRINT
 80 PRINTTAB(11)"** Worm Hole **"
 90 FOR N=92 TO 104
100 PRINT@N,"*";
110 FOR N1=1 TO 10:NEXT N1
120 NEXT N
130 FOR N=1 TO 500:NEXT N
140 FOR N=92 TO 104
150 PRINT@N,MID$(T$,N-91,1);
160 FOR N1=1 TO 100:NEXT N1
```

```
170 NEXT N
180 PRINT:PRINT:PRINTTAB(15)"(Y/N)"
190 A$=INKEY$:IF A$=""GOTO190
200 IF A$="Y" OR A$="y" GOTO210ELSE GOTO270
210 CLS:PRINT:PRINT
220 PRINTTAB(2)"Use arrow keys to avoid obstacles"
230 PRINTTAB(2)"and stay alive as long as possible."
240 PRINT:PRINT
250 PRINTTAB(10)"== Hit any key =="
260 A$=INKEY$:IF A$=""GOTO260

265 ' *** Set parameters ***

270 CLS:PRINT:PRINT
280 PRINTTAB(8)"Enter speed desired:"
290 PRINT
300 PRINTTAB(6)"[1] (Fast) to [9] (Slow)"
310 A$=INKEY$:IF A$=""GOTO310
320 DL=VAL(A$)*15
330 CLS:PRINT:PRINT
340 PRINTTAB(8)"Enter difficulty desired:"
350 PRINT:PRINTTAB(8)"[1] (Easy) to [9] (Hard)"
360 A$=INKEY$:IF A$=""GOTO360
370 D=VAL(A$)
380 DELTA=1

385 ' *** Randomize, and set up screen ***

390 FOR N=1 TO VAL(RIGHT$(TIME$,2))
400 DU=RND(1)
410 NEXT N
420 B1=1
430 C$="*"
440 CLS
450 FOR N=1 TO D*2.5
460 PRINT@RND(1)*320,CHR$(239);
470 NEXT N

475 ' *** Begin Play ***

480 A$=INKEY$
```

```
490 B1=B1+DELTA
500 IF B1<1 THEN B1=B1+DELTA
510 IF B1>320 THEN B1=B1-DELTA
520 IF PEEK(B1-512)<>32 GOTO650
530 PRINT @B1,C$;
540 PRINT@RND(1)*300,CHR$(32);
550 FOR N=1 TO DL:NEXT N
560 PT=PT+1
570 IF A$=""GOTO480
580 A=ASC(A$)
590 IF A<28 OR A>31 GOTO570
600 ON A-27 GOTO610,620,630,640

605 ' *** Move right,left, up or down ***

610 DELTA=+1:GOTO480
620 DELTA=-1:GOTO480
630 DELTA=-40:GOTO480
640 DELTA=40:GOTO480

645 ' *** Crash! ***

650 CLS:PRINT:PRINT
660 PRINTTAB(10)"You crashed!!"
670 PRINT:PRINTTAB(4)"You finished with "PT;" points."
680 PRINT:PRINTTAB(11)"Play again?"
690 PRINT:PRINTTAB(14)"(Y/N)"
700 A$=INKEY$:IF A$=""GOTO700
710 IF A$="Y" OR A$="y" THEN RUN
```

Chapter 9



Sketch

CAD, which can stand for computer assisted design or computer assisted drafting, is changing the way architects, engineers, and designers work. Instead of laboriously drafting on paper, professionals manipulate images on a CRT screen.

The Model 100 can also be used to draw images on its liquid crystal display screen. However, because this is a games book rather than an applications book, I have carefully designed the program Sketch so that it has no practical application.

There are a number of ways of drawing on the Model 100 screen. Up until this point, you have used only the print and print@ statements to place alphanumerics and graphics on the screen. When print@ statements are followed by semicolons, the screen does not scroll. Therefore, it is possible to move objects around on the screen quite quickly. If the previous position is erased after moving the object, the appearance is that the object has moved from one place to the other.

The problem with print@drawing is the lack of resolution. There are 320 print@ positions on the screen, making up a matrix that is 40 picture elements wide and eight picture elements deep. This gives new meaning to the term *low-res.*

However, the Model 100 does have another mode, which allows turning on or off any of the 15360 individual liquid crystal blocks, or pixels, on the screen. This array measures a more usable 240 across by 64 deep. For those of you familiar with the TRS-80 Models I/III and 4 (in Model III mode), the Model 100's resolution is roughly 2½ times better, although the aspect ratio is different.

These pixels are turned on using the PSET statement, and turned off via PRESET. Simply use the X and Y coordinates of the rectangle you want to switch, for example, PSET(11,23). Variables may also be substituted. One could draw a line thus:

```
10 FOR N=1 to 10:PSET(10,N):
   NEXT N
```

Yet another drawing mode can be accessed, using the line command. Here, two sets of coordinates must be entered, representing the starting point and ending point of the line. LINE (20,20)-(30,30) will draw a diagonal line on the screen. LINE (20,20)-(20-40) will draw a horizontal line, while LINE (20-20)-(30-20) will etch a vertical line on the screen.

Line gets even more interesting, because it can be followed by a switch. Odd switches (or no switch) tells BASIC to set the pixels in the line, while even values, for example, LINE (20-20)-(30,30),2, reset the pixels or turn them off.

But, as the TV pitchman says, wait! There's more! You also get to use a second set of switches, B and BF. The first draws a box with corners at the coordinates given. The second, BF fills in all the points inside that box.

Sketch uses both the PSET and line statements for its drawing routines. It has, in fact, two modes. The first, the sketch mode, uses only PSET. The cursor moves around the screen, leaving a trail of turned on pixels. If you wish, the cursor can be switched into the PRESET mode by touching the space bar. Then, although the cursor will still be visible, it will leave blank pixels wherever it goes. It can be turned back on by touching the space bar again. The space bar serves as what is called a *toggle* in computer slang. Pressing it forces the opposite of whatever mode the program happens to be in.

Sketch mode turns the Model 100 into the world's most expensive Etch-a-Sketch. The cursor direction is controlled by the arrow keys, and its speed can be changed at will, within limits. An initial speed is set by the operator when the program is run. The cursor can be slowed down from this speed or returned to it at any time, but it can never exceed the speed initially requested. Slowing is accomplished by pressing a number key, with the larger numbers providing more delay. Pressing the F key returns the cursor to full speed.

Draft mode allows the operator to set one point, and when a second point is set, a line is drawn between them. The program then pauses, awaiting input of either S, which sets the first point of the next line, or an arrow key, which starts the cursor moving in a new or the same direction. If you have drawn a line and want the new line to commence at the end of the old line, you press S before starting off. If you would prefer to start the new line at some other point, hit an arrow key instead. Then, press S at the first point of the new line. The next time S is pressed, the line will be drawn automatically.

Sketch begins with the player entering the length of delay desired. This value is multiplied by ten to produce a delay of 10 to 90. Variable OG keeps track of the length of delay originally set (line 320), so that the program can return to this as necessary. The actual delay used in the for-next loop is defined as DL, and can be changed in value during program execution.

If Sketch mode is selected, the program directs control to line 490, where an INKEY$ loop begins. Each time through the loop, the X and Y coordinates of the next pixel to be set or reset may be altered, depending on which arrow key was pressed last. The variables storing the X and Y coordinates have been given the names XD and YD. If the right or left arrow keys were pressed last, then XD will have a value of 1 or −1, respectively. YD will equal zero.

If the up or down arrow keys were pressed last, then XD will equal zero, and YD will be either −1 or 1. Thus, each time through the INKEY$ loop, the location of the next pixel to be turned on or off will be adjusted. Checks are made in lines 500 and 520 to make sure that movement is not going off the Model 100's screen.

Actual setting or resetting is done in lines 560 or 570, depending on the value of the variable FLAG. This variable is set to 1 each time the user switches from cursor-off mode to cursor-on mode. When FLAG=1, the program skips line 560, and

A$	Used in INKEY$ loop
A	Value of A$
DL	Current delay value
DM	Dummy variable for RND(1)
FLAG	Flag showing ON-OFF condition
N	Loop counter
OG	Original delay value
PO	Shows how many points have been set
X	Maximum movement allowed in X axis
X1	X axis position of cursor
X2	X axis position of first point set
Y	Maximum movement allowed in Y axis
Y1	Y axis position of cursor
Y2	Y axis position of first point set
YD	Y axis movement
XD	X axis movement

Fig. 9-1. Variables used in Sketch.

instead runs line 570, which turns the pixel on. When FLAG=0, line 560 first sets the pixel, so the user can see where the cursor is moving, waits during a for-next loop of 1 to 100, and then turns the pixel off before moving on.

All this happens between lines 490-580 when no key is pressed. When the user hits a key, control drops down to line 590, where a check is made to see if the key pressed was a number key. If so, the delay variable, DL, is multiplied by the value of the key pressed, thus increasing the delay. The program next checks to see if F was pressed. In that case, DL is returned to its original value of OG.

The program next looks to see if the space bar was pressed, changing the value of FLAG. Finally, the routine sifts out any input that wasn't an arrow key and allows the program to access one of four routines in lines 670-700 that change the X and Y coordinates appropriately.

Draft mode works similarly as far as cursor movement. However, no line is left by the moving cursor. Whenever the S key is pressed for the first time, that pixel is turned on to show the start position of the line to be drawn. That point is stored in X2 and Y2 as the coordinates to be used in the later line statement. When the second point in the line is set (PO=2), then the line is drawn in line 880, and the program waits for either another S or an arrow key before proceeding. A list of the variables used in the program is shown in Fig. 9-1.

Program Listing

```
10 ' ***********
20 ' *         *
30 ' * Sketch  *
40 ' *         *
50 ' ***********
```

```
 55 ' *** Instructions ***

 60 CLS:PRINT:PRINT
 70 PRINTTAB(12)"Instructions?"
 80 PRINT:PRINTTAB(16)"(Y/N)"
 90 A$=INKEY$:IF A$=""GOTO90
100 IF A$="Y" OR A$="y" GOTO 110 ELSE GOTO 270
110 CLS:PRINT:PRINT
120 PRINTTAB(2)"You may use either Sketch or Draft"
130 PRINTTAB(2)"modes.  Sketch uses a moving line to"
140 PRINTTAB(2)"draw, while Draft draws lines between"
150 PRINTTAB(2)"two points you specify by
    hitting";CHR$(34);"S";CHR$(34);"."
160 PRINT:PRINTTAB(1)"= Hit any key for Sketch instructions
    =";
170 A$=INKEY$:IF A$=""GOTO 170
180 CLS:PRINT
190 PRINTTAB(12)"==  Sketch =="
200 PRINTTAB(13)"[Controls:]"
210 PRINTTAB(2)"Arrow Keys -- Change direction"
220 PRINTTAB(2)"Number keys -- Slow down cursor"
230 PRINTTAB(2)CHR$(34);"F";CHR$(34);" -- Return to top
    speed"
240 PRINTTAB(2)"Space Bar -- Toggle cursor (ON)-(OFF)"
250 PRINT:PRINTTAB(11)"== Hit any key ==";
260 A$=INKEY$:IF A$=""GOTO 260

265 ' *** Set Delay ***

270 CLS:PRINT:PRINT
280 PRINTTAB(6)"Enter speed desired:"
290 PRINT
300 PRINTTAB(6)"[1] Fast to [9] Slow"
310 A$=INKEY$:IF A$=""GOTO310
320 OG=VAL(A$)*10:IF OG<1 GOTO310
330 DL=OG

335 ' ***  Choose Sketch or Draw Mode ***

340 CLS:PRINT:PRINT
350 PRINTTAB(4)"Do you want:"
```

```
360 PRINT:PRINTTAB(6)"[S]ketch mode"
370 PRINTTAB(6)"[D]raft mode"
380 A$=INKEY$:IF A$=""GOTO 380
390 IF A$="S"OR A$="s" GOTO 420
400 IF A$="D" OR A$="d"GOTO 710
410 GOTO 380
420 CLS
430 FOR N=1 TO VAL(RIGHT$(TIME$,2))
440 DM=RND(1)
450 NEXT N
460 X=239:Y=63
470 X1=INT(RND(X)*X)+1:Y1=INT(RND(X)*Y)+1
480 PSET(X1,Y1)

485 ' *** Sketch Mode ***

490 A$=INKEY$
500 X1=X1+XD:IF X1>239 THEN X1=239
510 IF X1<0 THEN X1=0
520 Y1=Y1+YD:IF Y1>63 THEN Y1=63
530 IF Y1<0 THEN Y1=0
540 FOR N=1 TO DL:NEXT N
550 IF FLAG<>1 GOTO570
560 PSET(X1,Y1):FORN=1 TO 100:NEXT N:PRESET(X1,Y1):GOTO580
570 PSET(X1,Y1)
580 IF A$=""GOTO490
590 IF VAL(A$)<1 GOTO610
600 DL=DL*VAL(A$)
610 IF A$="F" OR A$="f"THEN DL=OG
620 A=ASC(A$)
630 IF A<>32 GOTO650
640 IF FLAG=1 THEN FLAG=0 ELSE FLAG=1
650 IF A<28 OR A>31 GOTO490

655 ' *** Change direction of Cursor ***

660 ON A-27 GOTO670,680,690,700
670 YD=0:XD=1:GOTO490
680 YD=0:XD=-1:GOTO490
690 XD=0:YD=-1:GOTO490
700 YD=1:XD=0:GOTO490
```

```
 705 ' *** Draft  Mode ***

 710 X1=40:Y1=40
 720 CLS
 730 A$=INKEY$
 740 X1=X1+XD:IF X1>239 THEN X1=239
 750 IF X1<0 THEN X1=0
 760 Y1=Y1+YD:IF Y1>63 THEN Y1=63
 770 IF Y1<0 THEN Y1=0
 780 FOR N=1 TO DL:NEXT N
 790 PSET(X1,Y1)
 800 FOR N=1 TO 50:NEXT N
 810 PRESET(X1,Y1)
 820 IF A$=""GOTO730
 830 IF A$="S" OR A$="s" GOTO 850
 840 GOTO 990

 845 ' *** Set one point ***

 850 PO=PO+1
 860 PSET(X1,Y1)
 870 IF PO=1 THEN X2=X1:Y2=Y1:GOTO 730

 875 ' ***  Draw Line ***

 880 LINE (X2,Y2)-(X1,Y1)
 890 PO=0
 900 A$=INKEY$:IF A$=""GOTO 900
 910 IF A$="S" OR A$="s" GOTO 950
 920 A=ASC(A$)
 930 IF A<28 OR A>31 GOTO 900
 940 GOTO 980
 950 PO=1
 960 X2=X1:Y2=Y1
 970 GOTO 900

 975 ' *** Change Cursor Direction ***

 980 ON A-27 GOTO 990,1000,1010,1020
 990 YD=0:XD=1:GOTO730
1000 YD=0:XD=-1:GOTO730
1010 XD=0:YD=-1:GOTO730
1020 YD=1:XD=0:GOTO730
```

Chapter 10

Swap

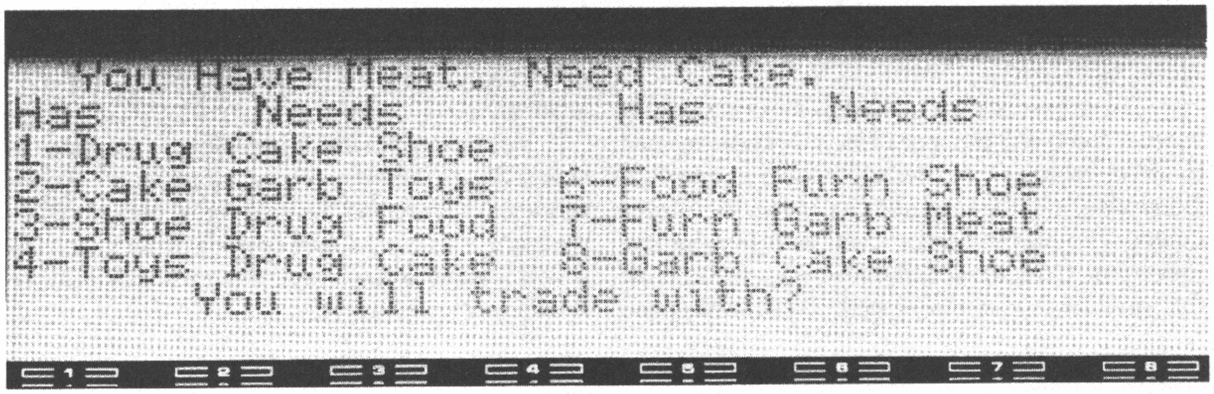
Simulations are a popular form of game for most computers, and the Model 100 is no exception. While wandering through CompuServe's Model 100 special interest group a few months after the computer was introduced, I found a lively group trying to adapt various adventure-style games to their portables. The reason for the interest is obvious. Unlike abstract arcade shoot-em-ups, simulations have some basis in real life. If the object of a game is to pretend to operate a candy store, to control the crops of a growing country, or to make stock market trades, the player can draw on his or her own knowledge or experience. Luck and chance can enter in, but logic and cunning also play a part.

Simulations allow computer users to learn a little about mathematical algorithms as well as the real world. In Swap, the object is to learn about bartering what you have for what you need in order to achieve your goal in as few transactions as possible.

At the beginning of each round, the player is assigned an occupation, such as druggist, baker, shoemaker, toymaker, butcher, grocer, cabinetmaker, or clothier. Each of these tradespeople have a ware to trade: drugs, cakes, shoes, toys, meat, food, furniture, and clothing. In the program, each product is abbreviated to four letters, if necessary, to allow correct screen formatting, as shown in Fig. 10-1.

The player will be assigned a need at random, while each of the other occupations in the round will have two needs of their own. The participant can trade what he or she has with any of the others who need one of those items.

The object is to trade what the player has for the target product in as few transactions as possible. For example, if the player is the druggist and his need is for toys, a direct trade could be made if the toymaker required drugs as one of his two needs. This will happen only infrequently. More often, the player will have to trade one product for another several times to arrive at something that



```
  You Have Meat. Need Cake.
Has    Needs          Has    Needs
1-Drug Cake Shoe
2-Cake Garb Toys  6-Food Furn Shoe
3-Shoe Drug Food  7-Furn Garb Meat
4-Toys Drug Cake  8-Garb Cake Shoe
      You will trade with?
```

Fig. 10-1. Screen display from Swap.

can be swapped for the target good.

The computer keeps track of the number of transactions needed to complete each round. To make this game more like the real world, I've resisted the urge to divide all the products up equally among all the business people as needs. If such were the case, solving each round would be little more than a math-like logic exercise. In Swap, there might be a heavy demand for meat with three or four of the players needing it, while only one might want shoes. If the player is acting as a butcher in that round, he will have many options in reaching his desired product—fewer if he is the shoemaker.

With this type of allocation, there will be some rounds in which it is impossible to get there from here. Then the player can quit the round and take 10 penalty points.

The player first enters the number of rounds to be played, which is stored in variable ROUND. A for-next loop from L to ROUND governs play. Each of eight products and occupations are read from data lines into two string arrays, PROD$(n) and OCCUP$(n). As each round begins, the player's identity is chosen (SELF) by selecting a random number between one and eight in line 450. Then, the extra product available, as well as the one needed by each of the other occupations in the round, are selected in lines 470 and 480. A check is made in lines 490-510 to make sure that the two products do not duplicate each other nor the product the occupation produces.

The transaction counter, TRANS, is incremented, and a list of the occupations (the player's excluded) is displayed. These are shown in two columns. A for-next loop from 1 to 4 prints OC-

A$	Used in INKEY$ loop
DU	Dummy variable for RND(1)
EXTRA$(n)	Extra product of given person
L	Loop counter of rounds
N	Loop counter
N1	Loop counter
N2	Loop counter
N3	Loop counter
N5	Loop counter
NEED$(n)	Item needed
NU	Person to trade with
OCCUP$(n)	Names of occupations
OTHER$(n)	Other need
PROD$(n)	Products of the occupations
ROUND	Number of rounds to be played
SC	Player's score
SELF	Occupation of the player
TRANS	Number of transactions completed so far
X	Random number
Y	Random number

Fig. 10-2. Variables used in Swap.

CUP$(N) as well as OCCUP$(N+4) on each line, along with the extra product (EXTRA$(N)) and need (NEED$(N)).

The player is then invited to make a trade. If winning the round is impossible, a Q can be entered to direct control to the default module at line 720. Otherwise, NU is assigned the value the player inputs. The player cannot trade with him/herself (line 760), and trading with those who do not need that product is also disallowed. To supervise the trade, the program compares what the player has (PROD$(SELF)) and the player's other product (OTHER$) with the NEED$(NU) of the occupation selected.

Each time a trade is completed, the player's score, SC, is incremented by the number of transactions, TRANS, and the next round played. When the game is over, the final score is printed, and the player invited to play again. The variables used are shown in Fig. 10-1.

Program Listing

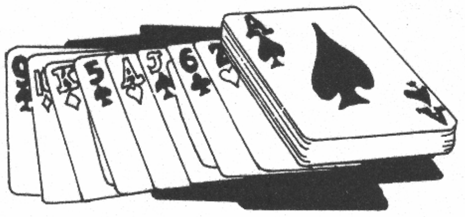
```
 10 ' *****************
 20 ' *               *
 30 ' *     Swap      *
 40 ' *               *
 50 ' *****************

 55 ' *** Set Random Start Point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N

 85 ' *** Instructions ***

 90 CLS:PRINT:PRINT
100 PRINTTAB(8)"DO YOU WANT INSTRUCTIONS?"
110 PRINT
120 PRINTTAB(16)"Y/N"
130 A$=INKEY$:IF A$="" GOTO 130
140 IF A$="Y" OR A$="y" GOTO 160
150 GOTO 350
160 CLS
170 PRINTTAB(2)"During this game, you will assume
180 PRINTTAB(2)"occupations,such as  doctor or farmer,
190 PRINTTAB(2)"and will attempt to trade your wares
200 PRINTTAB(2)"with others in your community so you
210 PRINTTAB(2)"can finish with the item you need.
220 PRINT
```

```
230 PRINTTAB(8)"== Hit any key =="
240 A$=INKEY$:IF A$="" GOTO 240
250 CLS
260 PRINTTAB(2)"You may need several trades to reach
270 PRINTTAB(2)"your goal.  The object is to trade
280 PRINTTAB(2)"efficiently and finish with as few
290 PRINTTAB(2)"transactions as possible.  If no one
300 PRINTTAB(2)"wants the goods you have, enter a 'Q'
310 PRINTTAB(2)"instead of a number to forfeit turn."
320 PRINT
330 PRINTTAB(8)"== Hit any key ==";
340 A$=INKEY$:IF A$="" GOTO 340
350 CLS
360 INPUT"How many rounds would  you like to play";ROUND$
370 ROUND=VAL(ROUND$)
380 DATA Drug,Cake,Shoe,Toys,Meat,Food,Furn,Garb
390 DATA Doctor,Baker,Cob'lr,Toyman,Butchr,Grocer,Carptr,
    Tailor
400 FOR N1=1 TO 8:READ PROD$(N1):NEXT N1
410 FOR N2=1 TO 8:READ OCCUP$(N2):NEXT N2

415 ' *** Start  Game ***

420 FOR L=1 TO ROUND
430 CLS:PRINT
440 OTHER$=""
450 SELF=INT(RND(1)*8)+1
460 FOR N3=1 TO 8
470 X=INT(RND(1)*8)+1
480 Y=INT(RND(1)*8)+1
490 IF X=Y GOTO 480
500 IF X=N3 GOTO 470
510 IF Y=N3 GOTO 480
520 EXTRA$(N3)=PROD$(Y)
530 NEED$(N3)=PROD$(X)
540 NEXT N3

545 ' *** Display Needs ***

550 CLS
560 IF OTHER$=EXTRA$(SELF) GOTO 920
570 TRANS=TRANS+1
```

```
580 PRINTTAB(2)"You Have ";PROD$(SELF);
590 IF OTHER$<>""THEN PRINT" and ";OTHER$;
600 PRINT".";
610 PRINT" Need ";EXTRA$(SELF);"."
620 PRINT"Has     Needs        Has    Needs"
630 FOR N5=1 TO 4
640 IF N5=SELF GOTO 680
650 PRINT MID$(STR$(N5),2,1);"-";
660 PRINT PROD$(N5);TAB(7)EXTRA$(N5);TAB(12);NEED$(N5);
670 IF N5+4=SELF THEN PRINT:GOTO 700
680 PRINTTAB(18);MID$(STR$(N5+4),2,1);"-";
690 PRINTPROD$(N5+4);TAB(25);EXTRA$(N5+4);
    TAB(30);NEED$(N5+4)
700 NEXT N5

705 ' *** Make Trade ***

710 PRINTTAB(6)"You will trade with?"
720 A$=INKEY$:IF A$="" GOTO 720
730 IF A$="Q" OR A$="q" GOTO 1150
740 NU=VAL(A$)
750 IF NU<1 GOTO 720
760 IF NU=SELF GOTO 720
770 IF PROD$(SELF)=EXTRA$(NU) OR OTHER$=EXTRA$(NU) GOTO 800
780 IF PROD$(SELF)=NEED$(NU) OR OTHER$=NEED$(NU) GOTO 830
790 GOTO 860
800 OTHER$=PROD$(NU)
810 EXTRA$(NU)=""
820 GOTO 550
830 OTHER$=PROD$(NU)
840 NEED$(NU)=""
850 GOTO 550

855 ' *** Invalid  Trade ***

860 CLS:PRINT:PRINTTAB(6)"Sorry.  But the ";OCCUP$(NU)
870 PRINTTAB(6)"does not need any ";PROD$(SELF);"."
880 PRINT
890 PRINTTAB(8)"== Hit any key =="
900 A$=INKEY$:IF A$="" GOTO 900
910 GOTO 550
```

```
 915 ' *** You won ***

 920 CLS:PRINT:PRINT
 930 PRINTTAB(2)"Congratulations."
 940 PRINTTAB(2)"You made the correct trades in "TRANS;"
 950 PRINTTAB(2)"transactions."
 960 SC=SC+TRANS
 970 PRINT:PRINT
 980 PRINTTAB(8)"== Hit any key =="
 990 A$=INKEY$:IF A$="" GOTO 990
1000 RESTORE
1010 TRANS=0
1020 NEXT L

1025 ' *** Game Over ***

1030 CLS
1040 PRINT
1050 IF SC <>0 THEN PRINTTAB(2)"Your score :";SC
1060 PRINT:PRINT
1070 PRINTTAB(6)"Play again?"
1080 A$=INKEY$:IF A$=""GOTO 1080
1090 IF A$="Y" OR A$="y" THEN RUN ELSE END

1145 ' *** Forfeit Round ***

1150 CLS:PRINT:PRINT
1160 PRINTTAB(2)"You forfeit turn and receive 10"
1170 PRINTTAB(2)"penalty points"
1180 SC=SC+10
1190 TRANS=0
1200 RESTORE
1210 PRINT
1220 PRINTTAB(8)"== Hit any key =="
1230 A$=INKEY$:IF A$="" GOTO 1230
1240 GOTO 1020
```

Chapter 11



Bettor Draw

Television game shows are good sources of ideas for computer games. Because new viewers have to comprehend how to play the game given only the host's 10 second explanation, TV game shows are usually simple in concept. Usually the nonskill shows require the participant only to guess what roll of the dice will come up next or what card will be drawn, or to spin a wheel of some sort. Most of us can handle that sort of action.

Bettor Draw happens to be played similarly to the bonus round of a now-defunct TV game show. There is no need to answer a difficult question or to name a tune in a minimal number of notes. Instead, the player has only to guess whether the next card drawn from a deck of 52 will be higher or lower than the previous card.

Trust me. This is not as difficult as it sounds. In this game, ace is low, and king is high, so when a king is drawn, the odds are very, very good that the next card will be lower. In fact, only drawing one of the other three kings can cause a player to lose. Conversely, an ace makes a good bet to stake everything you have, as only three cards in the deck will beat it. If another ace has been drawn previously in the round, your odds improve even more. So, it helps to remember what cards have already been played. Bettor Draw uses an accurate simulation of a deck of cards, even though we are only talking about a collection of electrons. A maximum of four of each value are available in any given game—honest.

As the cards drawn are close to the middle, 8, the odds become a bit more fuzzy. With a 10 showing, you will likely draw a lower card. But, a jack, queen, or king is a significant possibility. Hint: bet less than the max.

The player starts out with $200 and can bet all or part on each draw. Obviously, play depends on luck, along with some rudimentary knowledge of odds. Memory comes into play, too. If many face cards have been played, this would swing the odds toward the lower numbered cards.

Bettor Draw provides you with the opportunity to learn several exciting programming concepts, including how to simulate a deck of cards with the Model 100. Those of you who have noticed the many arrays used in this book may suspect that I will chose to have an array, with 52 elements, represent the deck of cards. That is correct. However, to understand how to deal them out, it may be best to look at the worst way first. Only then will you appreciate the clever programming trick that is commonly used.

Choosing a card should be done randomly. You could therefore choose a random number from 1 to 52 and select that as the next card dealt. That would work well except that there is no guarantee that the computer would not select that same card again on some other draw. Let's remove the card from the deck by causing that array element to become zero after the card is drawn:

```
10 DRAW=INT(RND(1)*52)+1

20 IF DECK(DRAW)=0 THEN
   GOTO 10

30 CARD=DECK(DRAW)

30 DECK(DRAW)=0

40 PRINT"Card drawn:";CARD

50 GOTO 10
```

With this module, the computer will select a card from the deck, but if it finds a zero in that position, it will loop back and try again. This is inefficient. In any game in which many cards are drawn, the computer will soon find itself discovering more zeros than cards. It may have to draw 20 or 30 times to find a card left.

The problem is that while in real life, the deck shrinks each time a card is drawn, your array remains the same size. Why don't you just make the array smaller each time a card is drawn?

```
10 NC=52

20 DRAW=INT(RND(1)*NC)+1

30 CARD=DECK(DRAW)

40 DECK(DRAW)=DECK(NC)

50 NC=NC-1

60 GOTO 20
```

This program will almost work. It starts off by setting NC (number of cards) to 52. In line 20, the computer selects a number between 1 and NC (52 this time), and that element of DECK(n) becomes the card drawn. This leaves a hole in the deck at position DRAW. You fill it up by taking the last card in the deck, which is DECK(NC), and placing it in DECK(DRAW). This leaves the hole at the end, but you then change NC to equal NC−1, so the computer will only draw from the elements 1 through 51 on the next time through. On the third time through, it will choose 1 through 50, and so forth. It does not matter that you have mixed up the order of the deck, as you want the cards shuffled in the first place.

This is basically what is done in Bettor Draw. The 52-element array is first loaded with the correct numbers, using a complex-looking set of nested for-next loops, beginning at line 280. The outer loop, from N=1 to 4, corresponds to the four suits. It repeats its operations four times to produce four identical suits, differing only in the graphic character used to produce the suit identifier. These are CHR$(156)−CHR$(159) in the Model 100, so SUIT$ becomes CHR$(155+N) each time through the outer loop. That string variable, SUIT$, is added to the numbers 1 through 10 to produce the

A$	Used in INKEY$ loop
BET	Player's bet
C	Random number, card drawn
CASH	Player's money
CH	Player's choice of higher or lower
CU	Counter, number of cards in suit created
DECK$(n)	Array, stores deck of cards
DU	Dummy variable for RND(1)
F$	Print using format
LC$	Last card drawn
N	Loop counter
N2	Loop counter
NC	Number of cards remaining in deck
SUIT$	Graphics character of current suit
V	Value of card drawn
V1	Value of card drawn
V2	Value of other card

Fig. 11-1. Variables used in Bettor Draw.

numbered cards. Then, the jack, queen, and king are added. Finally, the loop repeats to do the next suit.

Play begins at line 540, where a for-next loop counts off nine rounds. If CASH is less than 1, control goes to the bankrupt routine. Otherwise, the player's total winnings are printed, and the card most recently drawn (LC$, which will be the first card drawn on the first time through) is printed to the screen. The player may enter a guess of higher or lower. Then, the program branches to the choose-card subroutine at line 420.

Drawing of the card is done in a routine similar to the one described above. However, it is necessary to determine the value of the card. This is done in line 470 in the case of the numbered cards. You extract the value of face cards in lines 480-500 using INSTR. In lines 770-780, the values of the last card drawn and the next card drawn are compared to see if the player has made the right choice.

If he has, CASH is increased by BET, and the next round is played. If he hasn't, CASH is reduced by BET. Logical, eh? You will find a list of variables in Fig. 11-1. After nine variables rounds, the game is over. Who ways you don't have what it takes to go on television?

Program Listing

```
10 ' ***************
20 ' *             *
30 ' * Bettor Draw *
40 ' *             *
50 ' ***************
```

```
 55 ' *** Instructions ***

 60 CLS:PRINT:PRINT
 70 PRINTTAB(2)"Do you want instructions?"
 80 PRINT:PRINTTAB(14)"(Y/N)"
 90 A$=INKEY$:IF A$=""GOTO 90
100 IF A$="Y" OR A$="y" GOTO 120
110 GOTO 210
120 CLS:PRINT:PRINT
130 PRINTTAB(2)"Test your memory, and knowledge"
140 PRINTTAB(2)"of odds, while staking your"
150 PRINTTAB(2)"initial $200.   Each turn, you"
160 PRINTTAB(2)"can bet whether the next card "
170 PRINTTAB(2)"drawn will be higher or lower"
180 PRINTTAB(2)"than the last. You get nine rounds."
190 PRINTTAB(9)"== Hit any key =="
200 A$=INKEY$:IF A$="" GOTO 200
210 DIM DECK$(52)
220 F$="$$#####.##"
230 DECK=52

235 ' *** Set Random Starting Point ***

240 FOR N=1 TO VAL(RIGHT$(TIME$,2))
250 DU=RND(1)
260 NEXT N
270 CASH=200

275 ' *** Load Array with Deck ***

280 FOR N=1 TO 4
290   SUIT$=CHR$(155+N)
300     FOR N2=1 TO 10
310       CU=CU+1
320       DECK$(CU)=STR$(N2)+SUIT$
330     NEXT N2
340   CU=CU+1
350   DECK$(CU)="Jack "+SUIT$
360   CU=CU+1
370   DECK$(CU)="Queen "+SUIT$
380   CU=CU+1
```

```
390   DECK$(CU)="King "+SUIT$
400 NEXT N
410 GOTO 510

415 ' *** Choose Random Card ***

420 C=INT(RND(1)*DECK)+1
430 NC$=DECK$(C)
440 IF C=DECK GOTO 460
450 DECK$(C)=DECK$(DECK)
460 DECK=DECK-1
470 V=VAL(NC$):IF V>0 THEN RETURN
480 IF INSTR(NC$,"J")THEN V=11:RETURN
490 IF INSTR(NC$,"Q")THEN V=12:RETURN
500 IF INSTR(NC$,"K")THEN V=13:RETURN
510 GOSUB420
520 LC$=NC$
530 V1=V

535 ' *** Loop through nine rounds ***

540 FOR N=1 TO 9
550 IF CASH<1 GOTO 990
560 CLS
570 PRINT:PRINTTAB(5)"Total winnings:";
580 PRINT USING F$;CASH:PRINT
590 PRINTTAB(5) "Last drawn:";LC$
600 PRINT:PRINTTAB(5);"";:INPUT"Enter bet";BET
610 CLS:PRINT:PRINT
620 PRINTTAB(5)"Last drawn:"LC$:PRINT
630 PRINTTAB(5)" Will next be:"
640 PRINT
650 PRINTTAB(8)"[H]igher  or  [L]ower ?"
660 PRINT:PRINTTAB(8) "Enter choice:"
670 A$=INKEY$:IF A$=""GOTO 670
680 IF A$="H" OR A$="h"THEN CH=1:GOTO 710
690 IF A$="L"OR A$="l" THEN CH=0:GOTO 710
700 GOTO 670
710 GOSUB420
720 V2=V
730 CLS:PRINT:PRINT
```

```
740 PRINTTAB(5)"Last drawn:"LC$
750 PRINTTAB(5)"Next drawn:";NC$
760 PRINT
770 IF V2>V1 AND CH=1 GOTO 890
780 IF V2<V1 AND CH=0 GOTO 890
790 GOTO 940
800 LC$=NC$
810 V1=V2
820 NEXT N

825 ' *** End of Game Results ***

830 CLS:PRINT
840 PRINTTAB(4)"At end of game you have"
850 PRINT TAB(4);"";
860 PRINT USING F$;CASH
870 PRINT"left over."
880 GOTO 1010

885 ' *** Round won ***

890 PRINTTAB(10)"You win!"
900 CASH=CASH+BET
910 PRINT:PRINTTAB(2)"== Hit any key to play next round =="
920 A$=INKEY$:IF A$=""GOTO 920
930 GOTO 800

935 ' *** Round Lost ***

940 PRINTTAB(10)"You lose!"
950 CASH=CASH-BET
960 PRINT:PRINTTAB(2)"== Hit any key to play next round =="
970 A$=INKEY$:IF A$=""GOTO 970
980 GOTO 800

985 ' *** Player Bankrupt ***

 990 CLS:PRINT:PRINT
1000 PRINTTAB(5)" Sorry! You went bankrupt!"
1010 PRINT:PRINTTAB(9)"Another game? (Y/N)"
1020 A$=INKEY$:IF A$=""GOTO 1020
1030 IF A$="Y" OR A$="y"THEN RUN
```

Chapter 12

Duchy

Perhaps no one knows where the first Hamurabbi program originated. I certainly don't. Nevertheless, various Kingdom, Dukedom and Fiefdom programs of various sorts abound for most microcomputers. They are fairly simple to write, and you will find this one simple to understand. The hardest part is to think up a new royal-sounding name for the program. I chose Duchy. I think it has a concise-sound to it that becomes the compact Model 100.

The object is to become rich without starving everyone who lives on your land to death. Wealth is measured in the number of bushels of grain you have. Your title, which can range from master on through sir, duke, count, baron and prince to king determines what percentage of your serfs' crops you get to keep. Lowly masters get to keep only 20 percent. The rest is presumably passed on through higher echelons or hoarded by the dastardly serfs themselves. Counts, for example, get to keep 40 percent, while kings appropriate a confiscatory 95 percent of the harvest. Those kings really knew how to live.

You start the game as a master, and pass to higher titles as your land holdings increase. The first break comes at 10,000 acres, while a king must have 200,000 acres or more. Your title may fluctuate throughout the game as you buy and sell acreage. It may be wise to accept a lower share of the harvest in order to make a killing on the land market.

In fact, there are two ways to increase net worth in this game. One is through sowing and reaping: the farming game. This becomes more profitable as your title is enhanced and your share of the crop increased. Lower level royalty may make extra money by buying up land when it is cheap and selling it off when the price goes up. You have 20 rounds to reach the heights of wealth.

Your task is made more complex by the necessity to feed the people living on your land and to save enough grain for seed. Each man can till two acres. A woman can till one acre. And, since you are assuming that the scenario takes place before child labor laws went into effect, a child can till half an acre. If your sensibilities are offended, assume that the children are all strapping 17-year old males.

You may set aside 10 bushels a year to feed each person. If you allow less, some will starve. If you allow more, extra children will be born, and emigrants from other duchies will arrive in this land of plenty.

The program begins with an initialization of values, in lines 300-380. The first title bestowed is master, and this is stored in TITLE$. The amount of grain in store is set at somewhere between 1 and 999 bushels, and initial acreage is set at 1 to 9999 acres. Up to 249 people are established on your land, twice as many men as either women or children. The initial land price will be something between 1 and 11 bushels per acre.

Having these values change from game to game increases the challenge. In some games you may start out with 900 bushels of grain and 10,000 acres of land, immediately salable at 11 bushels an acre. This would be quite a head start. On the otherhand, you may start another game in the middle of a depression, with a bushel or two, a couple dozen acres, and 240 hungry mouths to feed. Not a pretty prospect.

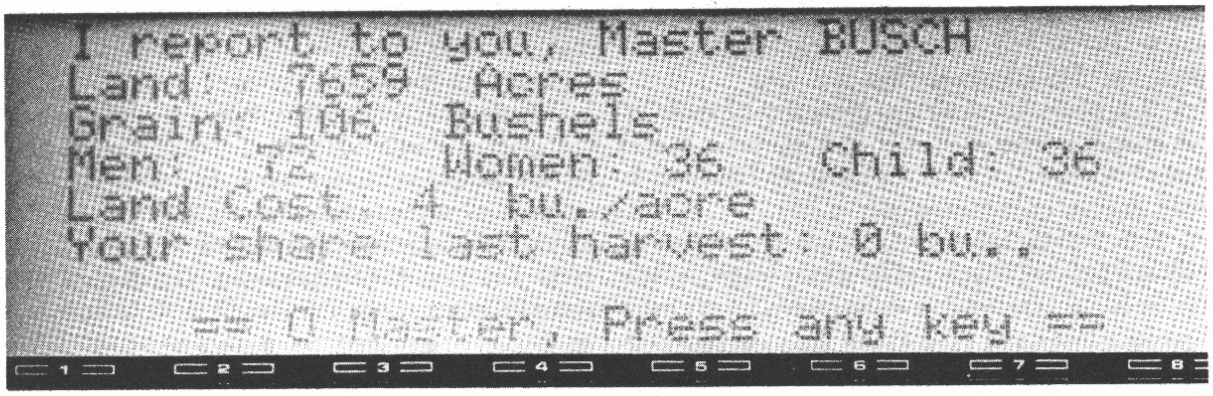
Actual game play commences at line 1250, where each round starts. A factor, FA, which can be either 1 or −1, determines whether land price will rise or fall each year. The amount of change, DELTA, is selected in line 1280 and can be as much as 11 bushels an acre per year.

After the new conditions are set, the results are displayed on the screen (Fig. 12-1). Then the player is allowed to buy acreage, sell land, set aside grain to feed his/her people, and plant. Each of these are taken care of in a separate subroutine.

At line 400, the amount of acreage is examined and the player's title set for the next round. Buying acreage is done beginning at line 490. You cannot buy more land than there is grain to pay for. Similarly, at lines 620-720, land can be sold and one's wealth (GRAIN) increased, but not at the expense of selling land one does not have. Bank loans and selling short are not implemented in this scenario.

The amount of grain set aside to feed people is taken care of beginning at line 1100. If less than the amount needed is provided or if more is set aside, a ratio is calculated that determines how many people will be left alive (or will arrive) in the Duchy the following year.

Finally, the player is allowed to plant. The number of acres tillable with the labor at hand is figured, using two acres for men, one for women,



```
I report to you, Master BUSCH
Land: 7659  Acres
Grain: 106  Bushels
Men: 72      Women: 36   Child: 36
Land Cost: 4  bu./acre
Your share last harvest: 0 bu..

    == O Master, Press any key ==
```

Fig. 12-1. Screen display from Duchy.

A$	Used in INKEY$ loop
ACRES	Number of acres of land owned
BUY	Number of acres to buy
CHILDREN	Number of children
DELTA	Change in land price
DU	Dummy variable for RND(1)
FA	Factor used to determine change
GRAIN	Bushels of grain in storage
HARVEST	Amount of grain harvested
LACK	Difference between grain fed and grain needed
LNDPRICE	Current land price
MEN	Number of men
N	Loop counter
NA$	Last name of player
NEEDED	Amount of grain needed to feed people
NT	Cost to buy acres
PEOPLE	Number of people
PLANT	Acres to plant
PP	Purchase price of land
RATS	Grain eaten by rats
SELL	Acres of land to be sold
SH	Player's percentage of harvest
TITLE$	Player's current title
WOMEN	Number of women
YEAR	Number of turns played so far

Fig. 12-2. Variables used in Duchy.

and half an acre for the strapping 17-year olds as a basis. Three bushels of grain per acre are needed as seed, and this is calculated in line 790.

The player is asked how many acres to sow. If not enough people are present, if there are fewer acres than input, or if not enough grain remains for seed, the entry is disallowed.

The harvest is more or less out of the player's hands. Up to 24 bushels per acre may result, which can be either very good or at the low end, very bad. Consider that three bushels per acre are required for seed, and 10 bushels per acre to feed the person tilling the land. Obviously, profit margins are very low. To make a dark picture even more pleasant, rats may eat some of the grain.

A player may lose all his/her land or all his/her grain, or both, without ill effect. Totally starving all his/her people is frowned upon, however, and will result in the game ending abruptly with the assassination of the player by friends of the dearly departed serfs. You should know that losing both all ones land and ones grain will directly result in this starvation on the very next turn. So, have a heart. A list of variables used in the program is found in Fig. 12-2.

Program Listing

```
 10 ' *********
 20 ' *       *
 30 ' * Duchy *
 40 ' *       *
 50 ' *********

 55 ' *** Set Random Start point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N

 85 ' *** Instructions ***

 90 CLS:PRINT
100 PRINTTAB(15)"Duchy"
110 PRINT:PRINTTAB(12)"Instructions?"
120 A$=INKEY$:IF A$=""GOTO 120
130 IF A$="Y" OR A$="y" GOTO 140 ELSE GOTO 290
140 CLS:PRINTTAB(2)"You will have 20 years (turns)"
150 PRINTTAB(2)"to rise from vassal to King.  You"
160 PRINTTAB(2)"may buy or sell land.  If you feed"
170 PRINTTAB(2)"your people well, more will be born"
180 PRINTTAB(2)"or emigrate.  Some may starve."
190 PRINT:PRINTTAB(10)"== Hit any key ==";
200 A$=INKEY$:IF A$=""GOTO 200
210 CLS:PRINTTAB(2)"Men can each till 2 acres"
220 PRINTTAB(2)"Women can till one acre. A child"
230 PRINTTAB(2)"may till just half an acre."
240 PRINTTAB(2)"You must loan your people seed"
250 PRINTTAB(2)"grain, which requires three"
260 PRINTTAB(2)"bu. an acre."
270 PRINT:PRINTTAB(8)"Hit any key to begin";
280 A$=INKEY$:IF A$=""GOTO 280

285 ' *** Initialize values ***

290 CLS
300 TITLE$="Master"
```

```
310 INPUT"Enter last name";NA$
320 GRAIN=INT(RND(1)*1000)
330 ACRES=INT(RND(1)*10000)
340 PEOPLE=INT(RND(1)*250)
350 MEN=INT(PEOPLE*.5)
360 WOMEN=INT(PEOPLE*.25)
370 CHILDREN=INT(PEOPLE-MEN-WOMEN)
380 LNDPRICE=INT(RND(1)*12)
390 GOTO 1250

395 ' *** Bestow New Title ***

400 IF ACRES>200000 THEN TITLE$="King": SHARE=.95: RETURN
410 IF ACRES>100000 THEN TITLE$="Prince": RETURN
420 IF ACRES>60000 THEN TITLE$="Baron": SHARE=.60: RETURN
430 IF ACRES>40000 THEN TITLE$="Count": SHARE=.4: RETURN
440 IF ACRES>20000 THEN TITLE$="Duke": SHARE=.3: RETURN
450 IF ACRES>10000 THEN TITLE$="Sir": SHARE=.25: RETURN
460 TITLE$="Master"
470 SHARE=.2
480 RETURN

485 ' *** Buy Acreage ***

490 CLS:PRINT
500 PRINT
510 PRINTTAB(2)"You currently have";GRAIN;"bu."
520 PRINTTAB(2)"of grain.  How many acres would"
530 PRINTTAB(2)"you like at ";LNDPRICE;" bu. per acre";
540 INPUT BUY
550 PP=BUY*LNDPRICE
560 NT=GRAIN-PP
570 IF NT<0 THEN  PRINTTAB(2)"Think again,";TITLE$;"
    ";NA$;", you have only ";GRAIN;" bu. of grain": GOTO 500
580 GRAIN=NT
590 ACRES=ACRES+BUY
600 GOSUB 400
610 RETURN

615 ' *** Sell Acreage ***

620 CLS:PRINT
```

```
630 PRINTTAB(2)"You currently own";ACRES
640 PRINTTAB(2)"acres of land.  How many to"
650 PRINTTAB(2)"sell at";LNDPRICE;"bu. per acre"
660 INPUT SELL
670 IF SELL>ACRES THEN PRINTTAB(2)"Think  again.  You can't
    sell land you do not have":FOR N=1 TO 500:NEXT N:CLS:
    GOTO 620
680 SOLD=SELL*LNDPRICE
690 GRAIN=GRAIN+SOLD
700 ACRES=ACRES-SELL
710 GOSUB 400
720 RETURN

725 ' *** Plant ***

730 CLS:PRINT
740 PRINT
750 PRINTTAB(2)"Your current population is";PEOPLE
760 TILLABLE=MEN*2
770 TILLABLE=TILLABLE+WOMEN
780 TILLABLE=TILLABLE+CHILDREN*.5
790 SEED=TILLABLE*3
800 SEED=INT(SEED):TILLABLE=INT(TILLABLE)
810 PRINTTAB(2)"They can till";TILLABLE;" acres if you have"
820 PRINTTAB(2)SEED;"bu. of grain left for seed."
830 PRINTTAB(2)"You have";GRAIN;"bu. in store."
840 PRINTTAB(2)"How many acres will you plant";
850 INPUT PLANT
860  IF PLANT>ACRES  THEN CLS:PRINTTAB(2)"You  haven't  that
    many acres!":GOTO810
870 IF PLANT>TILLABLE THEN CLS:PRINTTAB(2)"You don't have
    enough people to":PRINTTAB(2)" plant that much.": GOTO
    810
880 IF PLANT*3>GRAIN  THEN CLS:PRINTTAB(2)"You don't  have
    enough grain to":PRINTTAB(2)"plant that much.":GOTO 810
890 SEED=PLANT*3
900 GRAIN=GRAIN-SEED
910 RETURN

915 ' *** Harvest ***

920 FA=INT(RND(1)*24)
```

```
 930 HARVEST=PLANT*FA
 940 CLS:PRINT
 950 RATS=INT(RND(1)*11)
 960 RATS=HARVEST*RATS/10
 970 RATS=INT(RATS)
 980 HARVEST=HARVEST-RATS
 990 NT=HARVEST*SHARE
1000 NT=INT(NT)
1010 IF PLANT=0 THEN RETURN
1020 PRINTTAB(2)"Harvest was";FA;"bu. per acre."
1030 PRINTTAB(2)"Rats ate";RATS
1040 PRINTTAB(2)"Your share was ";NT;" bu. plus the"
1050 PRINTTAB(2)SEED;"bu. you supplied for seed."
1060 PRINT
1070 PRINTTAB(10)"== Hit any key =="
1080 A$=INKEY$:IF A$="" GOTO 1080
1090 RETURN

1095 ' *** Feed People ***

1100 CLS:PRINT
1110 PRINTTAB(2)"You have";GRAIN;"bu., and";PEOPLE;"people."
1120 PRINTTAB(2)"Each needs 10 bu. per year. How"
1130 PRINTTAB(2)"many bu. allocated to feed them"
1140 INPUT FEED
1150 IF FEED>GRAIN GOTO 1140
1160 GRAIN=GRAIN-FEED
1170 RETURN
1180 NEEDED=PEOPLE*10
1190 LACK=FEED/NEEDED
1200 PEOPLE=INT(PEOPLE*LACK)
1210 MEN=INT(PEOPLE*.5):WOMEN=INT(PEOPLE*.25)
1220 CHILDREN=INT(PEOPLE-MEN-WOMEN)
1230 IF PEOPLE=0 GOTO 1480
1240 RETURN

1245 ' *** Start Turn ***

1250 CLS:YEAR=YEAR+1
1260 IF YEAR=20 GOTO 1510
1270 FA=INT(RND(1)*2):IF FA<>1 THEN FA=-1
```

```
1280 DELTA=INT(RND(1)*12)*FA
1290 LNDPRICE=LNDPRICE+DELTA
1300 IF LNDPRICE<1 THEN LNDPRICE=1
1310 PRINTTAB(2)"I report to you, "TITLE$;" ";NA$
1320 PRINTTAB(2)"Land: ";ACRES;" Acres"
1330 PRINTTAB(2)"Grain:";GRAIN;" Bushels"
1340 PRINTTAB(2)"Men: ";MEN;TAB(14)"Women:";WOMEN;
     TAB(26)"Child:";CHILDREN
1350 PRINTTAB(2)"Land Cost:";LNDPRICE;" bu./acre"
1360 PRINTTAB(2)"Your share last harvest:";NT;"bu.."
1370 PRINT
1380 PRINTTAB(6)"== O ";TITLE$;", Press any key ==";
1390 A$=INKEY$:IF A$="" GOTO 1390
1400 GOSUB490
1410 GOSUB 620
1420 GOSUB 1100
1430 GOSUB 730
1440 GOSUB 920
1450 GOSUB 1180
1460 GOSUB 400
1470 GOTO 1250

1475 ' *** Everybody dead ***

1480 CLS:PRINTTAB(2)"Tyrant! You starved all your people!"
1490 PRINTTAB(2)"You lose!"
1500 GOTO 1560

1505 ' *** End of game ***

1510 CLS:PRINT
1520 PRINTTAB(2)"After 20 years, you finish with"
1530 PRINT ACRES;"acres of land";PEOPLE;" subjects,"
1540 PRINT GRAIN;"bu. of grain, and the title"
1550 PRINTTAB(2)"of "TITLE$;"!"
1560 PRINT:PRINTTAB(2)"Play again?"
1570 A$=INKEY$:IF A$="" GOTO 1570
1580 IF A$="Y" OR A$="y" THEN RUN
```

Chapter 13

Star Void

Star Void is included as a program for younger players. It gives all you frustrated space travelers the opportunity to pilot a rebel fleet toward home base, while avoiding Imperial raiders, diminished fuel supplies, and limited stocks of ammunition.

The object is to travel the 1000 parsecs in as few turns as possible. A straight-line approach is not possible. Because Imperials will attack several times en route, it is advisable to stock up on fuel and photon torpedoes as supplies are used up. When confronted by the enemy, you can fight or flee. Both choices are costly in terms of fuel, torpedoes, and lives.

The fleet begins with 50 fighters; at least some must reach the rebel base or else you lose.

Each new turn begins at line 330. The screen clears, and the turn counter, TURN, is incremented by one. Then, the program checks to see if fuel (F), ammo (A), distance remaining to base (D), or number of ships (T) is less than one. If fuel or ships are completely gone, the game is over. Low ammo supplies are not fatal, unless an attack by the enemy occurs. You are advised to seek more torpedoes immediately. If the distance remaining to the base is less than one, the player has won.

If none of these conditions are true, the status is displayed on the screen, with the amount of fuel, number of ships, distance, and amount of ammo shown. The player is offered three choices: get fuel, fly toward base, or get ammo. Each option is handled by a separate subroutine.

A handy programming technique to point out here is the input routine that is used in most of the programs in this book. It is common practice to present the user with a *menu* such as the one described above. Items from menus can be selected by having the player press the first letter of the item's name, enter the whole choice, or enter a number that precedes the menu choice.

The method of having the player type in the whole name is rarely used, because a simple typing error could invalidate an otherwise correct entry.

Entering one character is popular, especially when a menu is accessed frequently. The player can easily memorize which letter triggers which menu choice because of the mnemonic connection. The following is a typical letter-oriented menu:

(L)oad
(S)ave
(E)xit
(C)ontinue

A problem could occur if two menu choices started with the same letter, and the programmer could not think of a convenient synonym that used another initial letter. In addition, such menus force the nontypist user to hunt around the keyboard for letters that may be widely separated.

Numeric menus, on the other hand, have keys that are arranged in a neat row across the top of the keyboard. The player can also use the embedded numeric keypad of the Model 100. The limitation is that only ten menu choices can be listed, if you want single-key entry (0-9). Even there, you open yourself to problems, because the simplest input methods could confuse a null entry (just pressing the enter key, for example) with zero. It is possible to check the CHR$ values of the entries, to differentiate between zero (CHR$(48)) and enter (CHR$(13)). You could also extend a numeric menu by using hexadecimal notation, following 9 with A, B,C,D, or E.

In practice, this is seldom needed, especially with the Model 100. Since the portable has only an eight line screen, most menus will have fewer than eight choices. Even double column menus would probably have 12 or fewer choices to allow a line or two for a prompt.

Star Void's menus are of the numeric type. In addition, they have a built-in error trap, an item that is too often forgotten by beginning programmers. Some will write a routine like this:

```
10 PRINT"1.) Load program"

20 PRINT"2.) Save program"

30 INPUT"Enter Choice";CH

40 ON CH GOTO 100,200
```

Now, if a naive user enters L or S, or some other letter by mistake, a cryptic REDO FROM START message will be displayed. That is of no help at all. Entering a number larger than 2 will send the program to the line following 40, whatever that is. This could crash the whole program. You can avoid the redo message by using CH$ instead of CH in the input, since strings will accept letters as well as numbers. Converting to numerics, e.g., CH=VAL(CH$) will send you to our ON CH GOTO line happily—except you still haven't handled the inappropriate input that might result. Also, it is necessary for the user to remember to press the enter key before the input is accepted. The player either has to be sophisticated enough to do this on his or her own, or else you have to waste one of the Model 100's precious eight lines to prompt the player to do so.

Since all you want is a single character, why not use INKEY$ to get it? Then, if the character is not valid, just send control back to the INKEY$ loop until a proper entry is made. That is what is used in Star Void. Line 480, for example, is an INKEY$ loop that repeats until a character is pressed. That character, A$, is converted to a number value, CH, in line 490. If CH<1 or CH>3, the program loops back. Otherwise, it accesses the necessary subroutines.

To give Star Void some variety, several subroutines are accessed within routines. For example, when the player chooses to get fuel, a random number in the range 1 to 5 is chosen and one of five routines selected. These either renew the fuel, supply half a tank, no fuel, or some odd number of

A$	Used in INKEY$ loop
C	Random number
CH	Player's choice of action
D	Distance to base
DU	Dummy variable for RND(1)
F	Fuel remaining
H	Random number
HIGH	High score
I	Number torpedoes used up in action
J	Random number
N	Loop counter
NM$	Name of player
Q	Random number
SC	Random number
T	Fighters remaining
TURN	Current turn
U	Random number
V	Number torpedoes fired accidentally
V2	Fuel units used up in action
VU	Ships lost in action
W	Fuel obtained before flight

Fig. 13-1. Variables used in Star Void.

units (themselves selected from a random number less than 800).

Going to hyperspace prints a bunch of asterisks to the screen, in an attempt to simulate multimillion dollar special effects at very low cost. Four other subroutines produce various movements toward or away from the rebel base, dreaded visits to radiation areas, and an attack by the Imperial forces. The enemy attack is itself another routine with several subroutines. Gets complicated, doesn't it? Similarly, going to get ammo is a scenario with several outcomes determined by random factors.

A win occurs when the rebel base is reached with more than zero ships. If the number of turns is less than the previous high, a new record (yes, that is redundant) is established, and the player is allowed to play again.

Several death scenarios are also printed out beginning at line 2110, and the player is still allowed to try again. Figure 13-1 shows the variables used in the program.

Program Listing

```
10 ' *****************
20 ' *               *
30 ' *   STAR VOID   *
40 ' *               *
50 ' *****************
```

```
 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N
 90 HIGH=1000
100 CLS:PRINT:PRINT
110 PRINTTAB(12)"STAR VOID"
120 PRINT
130 PRINTTAB(4)"PLEASE ENTER YOUR NAME"
140 PRINT
150 INPUT NME$
160 PRINT
170 PRINTTAB(8)"INSTRUCTIONS (Y/N) ?"
180 A$=INKEY$:IF A$=""GOTO 180
190 IF A$="N" OR A$="n"GOTO 310
200 IF A$="Y" OR A$="y" GOTO 220
210 GOTO 180

215 ' *** INSTRUCTIONS ***

220 CLS
230 PRINT"You are 1000 parsecs from rebel base"
240 PRINT"with 50 fighters, 1000 fuel units and"
250 PRINT"1000 photon torpedoes. Imperials may"
260 PRINT"attack before you reach home.  If you"
270 PRINT"run out of ammo,fuel,or ships,you lose!"
280 PRINT
290 PRINTTAB(12)"== HIT ANY KEY =="
300 A$=INKEY$:IF A$=""GOTO 300
310 F=1000:T=50
320 D=1000:A=D

325 ' *** NEW TURN ***

330 CLS
340 TURN=TURN+1
350 IF F<1 THEN 2160
360 IF A<1 THEN 1250
370 IF D<1 THEN 2040
380 IF T<1 THEN 2140
390 V=0:C=0:N=0
400 PRINTTAB(4)"FUEL:";F
```

```
410 PRINTTAB(4)"SHIPS:";T
420 PRINTTAB(4)"DISTANCE:";D
430 PRINTTAB(4)"TORPEDOES:";A
440 PRINTTAB(8)"DO YOU WANT TO"
450 PRINTTAB(6)"1.) GET FUEL"
460 PRINTTAB(6)"2.) FLY TOWARD BASE"
470 PRINTTAB(6)"3.) GET AMMO";
480 A$=INKEY$:IF A$=""GOTO 480
490 CH=VAL(A$)
500 IF CH<1 OR CH>3 GOTO 480
510 ON CH GOTO 520,850,1270

515 ' *** GET FUEL ***

520 CLS:PRINT:PRINT
530 PRINTTAB(2)"WE'RE GOING FOR FUEL. SCANNERS"
540 PRINTTAB(2)"LOOKING FOR NEAREST FUEL DEPOT."
550 PRINTTAB(2)NME$;"!"
560 FOR N=1 TO 2000
570 NEXT N
580 C=INT(RND(1)*4)+1
590 ON C GOTO 710,650,770,600,600
600 CLS:PRINT: PRINTTAB(2)"YIPES! DEPOT EMPTY!"
610 PRINTTAB(2)"WE STILL HAVE";F;"UNITS OF FUEL."
620 PRINTTAB(12)"== HIT ANY KEY =="
630 A$=INKEY$:IF A$=""GOTO 630
640 GOTO 330
650 F=F+1000
660 CLS:PRINT:PRINTTAB(2)"WE FILLED UP WITH FUEL.  WE NOW
    HAVE"
670 PRINT"";F;"UNITS."
680 PRINTTAB(12)"== HIT ANY KEY =="
690 A$=INKEY$:IF A$=""GOTO 630
700 GOTO 330
710 F=F+500
720 CLS:PRINT:PRINTTAB(2)"THE DEPOT COULD ONLY FILL US
    HALFWAY."
730 PRINTTAB(2)"WE NOW HAVE";F;"UNITS OF FUEL."
740 PRINTTAB(12)"== HIT ANY KEY =="
750 A$=INKEY$:IF A$=""GOTO 630
760 GOTO 330
770 W=INT(RND(1)*800)
```

```
 780 F=F+W
 790 CLS:PRINT:PRINTTAB(2)"IMPERIAL FIGHTERS ARE COMING!"
 800 PRINTTAB(2)"WE HAD TO LEAVE BEFORE WE COULD FILL"
 810 PRINTTAB(2)"UP.  WE DID GET ";F;"UNITS OF FUEL."
 820 PRINTTAB(12)"== HIT ANY KEY =="
 830 A$=INKEY$:IF A$=""GOTO 630
 840 GOTO 330

 845 ' *** GO BASE ***

 850 CLS:PRINT
 860 PRINTTAB(2)"WE'RE GOING INTO HYPERSPACE. HANG ON."
 870 FOR N=1 TO 500:NEXT N
 880 FOR N=1 TO 200
 890 PRINT":*:";
 900 NEXT N
 910 CLS:PRINT
 920 H=INT(RND(1)*3)+1
 930 IF H=2 THEN 1620
 940 C=INT(RND(1)*4)+1
 950 U=INT(RND(1)*500)+1
 960 ON C GOTO 1020,1080,1150,970
 970 U=INT(U/10)
 980 CLS:PRINTTAB(2)"YOU WERE CAUGHT IN A SPACE STORM AND"
 990 PRINTTAB(2)"SENT ";U;"PARSECS OFF COURSE."
1000 D=D+U
1010 GOTO 1100
1020 U=INT(U*1.5)
1030 D=D-U
1040 CLS:PRINT:PRINTTAB(2)"YOU WENT";U;"PARSECS"
1050 PRINTTAB(2)"TOWARD THE BASE.  YOU ARE NOW ";D
1060 PRINTTAB(2)"PARSECS AWAY."
1070 GOTO 1100
1080 CLS:PRINT:PRINTTAB(2)"YOU GOT STALLED IN A TIME WARP,
     AND GOT"
1090 PRINTTAB(2)"NOWHERE."
1100 F=F-U
1110 PRINTTAB(2)"YOU USED UP";U;"UNITS OF FUEL"
1120 PRINTTAB(12)"== HIT ANY KEY =="
1130 A$=INKEY$:IF A$=""GOTO 630
1140 GOTO 330
```

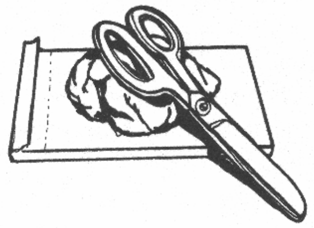
```
1150 CLS:PRINT
1160 PRINTTAB(2)"YOU ENTERED A RADIATION AREA.  IT"
1170 V=INT(U/2)
1180 PRINTTAB(2)"CAUSED";V;"PHOTON TORPEDOES TO FIRE"
1190 PRINTTAB(2)"LUCKY, NONE OF YOUR SHIPS WERE DESTROYED."
1200 A=A-V
1210 PRINTTAB(2)"YOU HAVE";A;"UNITS OF AMMO LEFT."
1220 PRINTTAB(12)"== HIT ANY KEY =="
1230 A$=INKEY$:IF A$=""GOTO 1230
1240 GOTO 330
1250 CLS:PRINT:PRINTTAB(2)"OUT OF AMMO! GO GET MORE!"
1260 GOTO 390

1265 ' *** GET AMMO ***

1270 CLS:PRINT
1280 PRINTTAB(2)"LOOKING FOR AN AMMO SUPPLY DEPOT."
1290 FOR N=1 TO 1000
1300 NEXT N
1310 C=INT(RND(1)*6)+1
1320 Q=INT(RND(1)*500)+1
1330 IF C=1 THEN GOTO 1410
1340 IF C=2 THEN GOTO 1470
1350 A=A+Q
1360 CLS:PRINT:PRINTTAB(2)"DEPOT GAVE US";Q;"PHOTON
     TORPEDOES."
1370 PRINTTAB(2)"WE NOW HAVE";A
1380 PRINTTAB(12)"== HIT ANY KEY =="
1390 A$=INKEY$:IF A$=""GOTO 1390
1400 GOTO 1560
1410 CLS:PRINT:PRINTTAB(2)"NO AMMO AVAILABLE"
1420 PRINTTAB(2)"YOU STILL HAVE";A;"TORPEDOES."
1430 PRINTTAB(12)"== HIT ANY KEY =="
1440 A$=INKEY$:IF A$=""GOTO 1390
1450 GOTO 1560
1460 Q=INT(Q/4)
1470 CLS:PRINT:PRINTTAB(2)"YIPES! YOU WERE ATTACKED BY
     IMPERIALS"
1480 PRINTTAB(2)"BEFORE YOU COULD GET TO THE DEPOT."
1490 IF A<1 THEN GOTO 2110
1500 PRINTTAB(2)"YOU USED UP ";Q;"TORPEDOES TO ESCAPE!"
```

```
1510 IF A>0 THEN 1530
1520 PRINTTAB(2)"YOU USED UP ALL OF YOUR AMMO!"
1530 A=A-Q
1540 PRINTTAB(12)"== HIT ANY KEY =="
1550 A$=INKEY$:IF A$=""GOTO 1550
1560 CLS:PRINT:PRINTTAB(2)"YOU USED UP ";Q;"UNITS OF FUEL"
1570 PRINTTAB(2)"LOOKING FOR AMMO"
1580 PRINTTAB(12)"== HIT ANY KEY =="
1590 A$=INKEY$:IF A$=""GOTO 1590
1600 F=F-Q
1610 GOTO 330

1615 ' *** IMPERIAL FIGHTERS ATTACKING ***

1620 CLS:PRINT:PRINTTAB(2)"IMPERIAL FIGHTERS ATTACKING"
1630 IF A<1 THEN OTO 2670
1640 PRINTTAB(8)"MAKE CHOICE:"
1650 PRINTTAB(12) "1.) ATTACK"
1660 PRINTTAB(12)"2.) FLEE"
1670 A$=INKEY$:IF A$=""GOTO 1670
1680 V=VAL(A$)
1690 IF V<1 OR V>2 GOTO 1670
1700 IF V=2 THEN GOTO 1860
1710 PRINT:PRINTTAB(14)"ATTACKING.."
1720 FOR N=1 TO 2000:NEXT N
1730 FOR N=1 TO 50
1740 SC=INT(RND(1)*319)+1
1750 PRINT@SC,"";
1760 NEXT N
1770 VU=INT(RND(1)*6)+1
1780 CLS:PRINT:PRINTTAB(2)"YOU WON, BUT YOU LOST";VU;
     "SHIPS AND"
1790 I=VU*100
1800 PRINTTAB(2)"USED UP";I;"TORPEDOES."
1810 A=A-I
1820 T=T-VU
1830 PRINTTAB(12)"== HIT ANY KEY =="
1840 A$=INKEY$:IF A$=""GOTO 1840
1850 GOTO 330
1860 PRINT:PRINTTAB(2)"ESCAPING.."
1870 FOR N=1 TO 2000:NEXT N
```

```
1880 V2=INT(RND(1)*200)+1
1890 PRINTTAB(2)"YOU USED UP";V2;"UNITS OF FUEL, BUT"
1900 J=INT(RND(1)*3)+1
1910 IF J=1 THEN GOTO 2000
1920 PRINTTAB(2)"YOU WERE ATTACKED ANYWAY.  YOU LOST"
1930 PRINTTAB(2)J;"SHIPS."
1940 PRINTTAB(2)"YOU USED UP";V2;"TORPEDOES."
1950 A=A-V2
1960 PRINTTAB(12)"== HIT ANY KEY =="
1970 A$=INKEY$:IF A$=""GOTO 1970
1980 T=T-J
1990 GOTO 330
2000 PRINTTAB(2)"YOU ESCAPED."
2010 PRINTTAB(12)"== HIT ANY KEY =="
2020 A$=INKEY$:IF A$=""GOTO 2020
2030 GOTO 330

2035 ' *** YOU WIN ***

2040 CLS:PRINT:PRINTTAB(2)"YOU HAVE REACHED BASE SAFELY WITH"
2050 PRINTTAB(2)T;"SHIPS LEFT!"
2060 PRINTTAB(2)"YOU TOOK ";TURNS;"TURNS."

2065 ' *** NEW RECORD? ***

2070 IF TURNS<HIGH THEN HIGH=TURNS:GOTO 2090
2080 GOTO 2170
2090 PRINTTAB(2)"NEW RECORD!"
2100 GOTO 2170

2105 ' *** YOU LOSE ***

2110 CLS:PRINT:PRINTTAB(2)"IMPERIAL FIGHTERS ATTACKED WHEN"
2120 PRINTTAB(2)"OUT OF AMMO. YOU ARE DEAD. YOU LOSE!!"
2130 GOTO 2170
2140 PRINTTAB(2)"ALL YOUR SHIPS ARE DESTROYED. YOU LOSE."
2150 GOTO 2170
2160 CLS:PRINT:PRINTTAB(2)"OUT OF FUEL. YOU ARE DEAD. YOU LOSE!"
2170 PRINT:PRINTTAB(8)"ANOTHER GAME?"
2180 A$=INKEY$:IF A$=""GOTO 2180
2190 IF A$="Y"OR A$="y" GOTO 310
```

Chapter 14

Paper, Rock, Scissors



Dungeons and Dragons games are very popular, due to the fantasy element wherein players assume the guises of wizards, warriors, trolls, and other people from normal walks of life. Because D & D games require a great deal of study of lore, martial arts, and other fields, I decided not to include one in this book. However, if you can imagine yourself as an anthropomorphic paper, a living rock, or a menacing magical pair of scissors, you can get some of the fun of Dungeons and Dragons from this game.

After all, like D & D, each character has its own set of attributes and hidden weaknesses. The mighty paper is capable of smothering some of its enemies, although it can be slashed to ribbons by the vicious scissors. The rock, while susceptible to attacks by paper, is capable of smashing scissors to death. All cut-and-dried, right?

The only problem with Paper, Rock, Scissors cum D & D is that it might be too violent for some. After all, you won't see smothering, hacking to pieces, or stoning to death on prime time television.

Although PRS is longer than some, the game is very simple. Most of the length comes from writing similar routines to take care of all the possible permutations of player and computer choices. The player selects first, inputting either P, R or S in line 310. The computer chooses next.

Computer neophytes may be a bit suspicious here. Allowing the computer to choose its weapon after the player has entered a choice leaves the game wide open for computer cheatery. Those who know better know better. The computer makes its selection in line 340, by taking a random number from 1 to 3. This selection has nothing at all to do with the value of A$. I know—I checked. Distrusting types can move line 340 up to before the player makes a choice. For this tactic to work, however, you will have to renumber the line. A line number like 305 would be appropriate. It will be necessary to change line 350's number to 340, however, because the program jumps to 340 from line 320. Aren't you ashamed for causing all that trouble?

Each of three major routines are built around

A$	Used in INKEY$ loop
CH	Computer choice
CP	Computer's points
DU	Dummy variable for RND(1)
N	Loop counter
PP	Player's points

Fig. 14-1. Variables used in Paper, Rock, and Scissors.

the computer's choice, e.g., paper, rock or scissors. Within each of those routines are three subroutines, based on the player's choice, which handle whether you won, lost, or tied. You get a point for winning; the computer gets a point when it wins. Nobody gets a point when both choose the same weapon.

That's all there is to it. The computer's points are stored in variable CP. The person's points are stored in variable PP. A list of the variables used is shown in Fig. 14-1. Whichever reaches 15 points is declared the winner.

Don't try using psychology on the computer, though. Thinking "It'll never guess I'm going for rock, because I went for rock three times in a row already!" is as good a strategy as any other, in any case.

Program Listing

```
 10 ' **************************
 20 ' *                        *
 30 ' * Paper, Rock, Scissors *
 40 ' *                        *
 50 ' **************************

 55 ' *** Instructions ***

 60 CLS:PRINT:PRINT
 70 PRINTTAB(2)"Do you want instructions?"
 80 PRINTTAB(12)"(Y/N)"
 90 A$=INKEY$:IF A$=""GOTO90
100 IF A$="Y" OR A$="y" GOTO120
110 GOTO170
120 CLS:PRINT:PRINT
130 PRINTTAB(2)"Try to beat the computer to"
140 PRINTTAB(2)"fifteen points by selecting"
150 PRINTTAB(2)"paper, rock, or scissors wisely."
160 PRINT:PRINTTAB(14)"Good luck!"

165 ' *** Set random seed ***

170 FOR N=1 TO VAL(RIGHT$(TIME$,2))
180 DUM=RND(1)
```

```
190 NEXT N
200 PRINT:GOTO220
210 CLS:PRINT:PRINT
220 PRINTTAB(5)"== Hit any key to continue. =="

225 ' *** Display Choices  ***

230 A$=INKEY$:IF A$=""GOTO230
240 IF CP>14 OR PP>14 GOTO1010
250 CLS:PRINT:PRINT
260 PRINT:PRINTTAB(5)"You:";PP;TAB(20);"Computer:";CP
270 PRINT:PRINTTAB(8)"[P]aper"
280 PRINTTAB(8)"[R]ock"
290 PRINTTAB(8)"[S]cissors"
300 PRINT:PRINTTAB(5)"Select:";
310 A$=INKEY$:IF A$=""GOTO310
320 IF A$="P"ORA$="p"ORA$="R"ORA$="r"OR A$="S"OR A$="s"
    GOTO340
330 GOTO310

335 ' *** Computer makes choice ***

340 CH=INT(RND(1)*3)+1
350 ON CH GOTO360,580,800

355 ' *** Computer chose Paper... ***

360 CLS:PRINT:PRINT
370 PRINTTAB(5)"Computer chose Paper"
380 PRINTTAB(5)"You chose ";
390 IF A$="P" OR A$="p" GOTO420
400 IF A$="R" OR A$="r" GOTO460
410 IF A$="S" OR A$="s" GOTO520

415 ' *** ...and Player chose Paper ***

420 PRINTTAB(5)"Paper."
430 PRINTTAB(5)"Tie. No points."
440 PRINT
450 GOTO220

455 ' *** ...and Player chose Rock ***
```

```
460 PRINTTAB(5)"Rock."
470 PRINTTAB(5)"Paper wraps Rock."
480 PRINTTAB(5)"Point for Computer."
490 CP=CP+1
500 PRINT
510 GOTO220

515 ' *** ...and Player chose Scissors ***

520 PRINTTAB(5)"Scissors."
530 PRINTTAB(5)"Scissors cut Paper."
540 PRINTTAB(5)"You get one point."
550 PRINT
560 PP=PP+1
570 GOTO220

575 ' *** Computer chose Rock... ***

580 CLS:PRINT:PRINT
590 PRINTTAB(5)"Computer chose Rock"
600 PRINTTAB(5)"You chose ";
610 IF A$="P" OR A$="p" GOTO640
620 IF A$="R" OR A$="r" GOTO700
630 IF A$="S" OR A$="s" GOTO740

635 ' *** ...and Player chose Paper ***

640 PRINT"Paper"
650 PRINTTAB(5)"Paper wraps Rock."
660 PRINTTAB(5)"You get one point."
670 PP=PP+1
680 PRINT
690 GOTO220

695 ' *** ...and Player chose Rock ***

700 PRINT"Rock"
710 PRINTTAB(5)"Tie. No points."
720 PRINT
730 GOTO220

735 ' *** ...and Player chose Scissors ***
```

```
740 PRINT"Scissors"
750 PRINTTAB(5)"Rock breaks Scissors."
760 PRINTTAB(5)"Computer gets one point."
770 CP=CP+1
780 PRINT
790 GOTO220

795 ' *** Computer chose Scissors... ***

800 CLS:PRINT:PRINT
810 PRINTTAB(5)"Computer chose Scissors"
820 PRINTTAB(5)"You chose ";
830 IF A$="P" OR A$="p" GOTO860
840 IF A$="R" OR A$="r" GOTO920
850 IF A$="S" OR A$="s" GOTO980

855 ' *** ...and Player chose Paper ***

860 PRINT"Paper."
870 PRINTTAB(5)"Scissors cut Paper"
880 PRINTTAB(5)"Point for computer."
890 CP=CP+1
900 PRINT
910 GOTO220

915 ' *** ...and Player chose Rock ***

920 PRINT"Rock."
930 PRINTTAB(5)"Rock breaks Scissors."
940 PRINTTAB(5)"Point for you."
950 PP=PP+1
960 PRINT
970 GOTO220

975 ' *** ...and Player chose Scissors ***

980 PRINT"Scissors."
990 PRINTTAB(5)"Tie. No points."
1000 GOTO220

1005 ' *** End of Game ***
```

```
1010 CLS:PRINT:PRINT
1020 PRINTTAB(2)"Fifteen points achieved!"
1030 IF CP>14 THEN PRINTTAB(2)"Computer wins!"
1040 IF PP>14 THEN PRINTTAB(2)"Human wins!
1050 PRINT:PRINT
1060 PRINTTAB(8)"Play again?"
1070 A$=INKEY$:IF A$=""GOTO1070
1080 IF A$="Y" OR A$="y" THEN RUN
1090 CLS
```

Chapter 15

Bingo

Ever get the urge to play Bingo with 10 or 15 business associates in the middle of a flight from Dallas to Miami? Now, with this program and the Model 100 computer, you can do it. You say you'd rather play Bingo in the off-hours with friends or family? You can do that, too, but I was looking forward to hearing "I-23!!" waft back from the first class section of the next flight I take.

Bingo will draw numbers at random, one each time you touch a key on the Model 100 keyboard. They will be displayed on the screen, gradually scrolling off. However, you can recall the entire list at any time by pressing either the up or down arrow keys.

Drawing the numbers is a lot like choosing the cards from the deck in Bettor Draw. The deck, or array, with 75 numbers, is a bit larger. That difference does not faze the Model 100, which, after all, is the only Radio Shack computer that has a three-digit number as part of its very name. The device can handle numbers that are even larger with aplomb.

The tricky part is matching the numbers from 1 to 75 with the corresponding letter in B,I,N,G,O. This is solved by listing these five letters in a data line, and then reading them into a string array, ROW$(n).

As you load the numbers 1-75 into the array BNG$(n), a check is made to see if the loop counter, N, is evenly divisible by 15. If so, a counter is incremented, moving one farther along in the array ROW$(n). In other words, until N=16, ROW$(1), or B is added to the number to produce B−1, B−2, etc. Once N is higher than 16, ROW$(2), or I is used.

There are lots of ways of determining which element of ROW$(n) to use. I chose this route in order to illustrate the MOD statement, a very useful device. In BASICS without MOD, you find out whether or not a number is evenly divisible by another by dividing it by that number, and comparing it to the INT of that some operation. Like so:

```
10 IF N/15=INT(N/15) THEN
```

```
PRINT"Evenly Divisible"
```

For example, if N=30, then N/15=2 and INT(N/15) also equals 2. However, if N=32, then N/15=2.13333333 and INT(N/15) still equals 2. With the Model 100, there is no need to go to that bother. Instead, IF N MOD 15=0 THEN PRINT"Evenly Divisible" will work. You've saved seven or so keystrokes and managed to impress a few people who are not in on the secret of modulus arithmetic.

Drawing numbers commences at line 340, where a handy INKEY$ loop awaits keyboard input. If A$ happens to equal CHR$(30) or CHR$(31) (the up and down arrow keys), the program branches to line 450, where all the numbers drawn so far are displayed.

Otherwise, a random number, R, is chosen from 1 to NC, which is the size of the array remaining. The drawn number, DRAW$, is extracted from BNG$(n), and the array closed up as described in Bettor Draw. This number is printed to the screen and stored in D$(n), an array that keeps track of all the numbers drawn so far. A counter, D, is incremented, to keep track of the next available position in D$(n).

When a review is required, a for-next loop from 1 to D repeats, printing each element of D$(n) in turn and pausing during a delay loop at line 480 before going on to the next element. A list of the varibles used in the program is shown in Fig. 15-1.

A$	Used in INKEY$ loop
BNG$(n)	Array storing undrawn numbers
CU	Counter used in loading array
D$(n)	Array storing drawn numbers
DU	Dummy variable for RND(1)
N	Loop counter
N2	Loop counter
NC	Number of undrawn numbers remaining
R	Number drawn at random
ROW$(n)	Array storing letters B,I,N,G,O

Fig. 15-1. Variables used in Bingo.

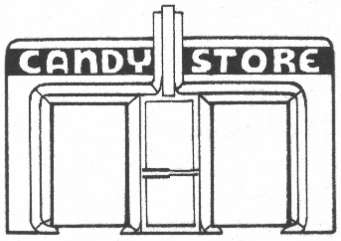
Program Listing

```
10 ' *********
20 ' *       *
30 ' * Bingo *
40 ' *       *
50 ' *********
60 CLEAR 1000
70 DIM BNG$(75),D$(75)

75 ' *** Set Random Start Point ***
```

```
 80 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 90 DU=RND(1)
100 NEXT N

105 ' *** Instructions ***

110 CLS:PRINT:PRINTTAB(12)"Instructions?"
120 PRINT:PRINTTAB(16)"Y/N"
130 A$=INKEY$:IF A$="" GOTO 130
140 IF A$="Y" OR A$="y" GOTO 150 ELSE GOTO 230
150 CLS:PRINT
160 PRINTTAB(2)"To draw a number, hit any key."
170 PRINTTAB(2)"To review numbers already drawn"
180 PRINTTAB(2)"press up or down arrow keys."
190 PRINT
200 PRINTTAB(6)"== Hit any key to begin =="
210 A$=INKEY$:IF A$="" GOTO 210

215 ' *** Read Bingo Array ***

220 DATA B,I,N,G,O
230 CU=1
240 NC=75
250 D=1
260 FOR N=1 TO 5
270 READ ROW$(N)
280 NEXT N
290 FOR N=1 TO 75
300 BNG$(N)=ROW$(CU)+"-"+MID$(STR$(N),2)
310 IF N MOD 15=0 THEN CU=CU+1
320 NEXT N
330 CLS

335 ' *** Start Drawing ***

340 A$=INKEY$:IF A$=""GOTO 340
350 IF A$=CHR$(30) OR A$=CHR$(31) GOTO 450
360 R=INT(RND(1)*NC)+1
370 DRAW$=BNG$(R)
380 IF R=NC GOTO 400
390 BNG$(R)=BNG$(NC)
```

```
400 NC=NC-1
410 PRINTTAB(16);DRAW$
420 D$(D)=DRAW$
430 D=D+1
440 GOTO 340

445 ' *** Review Drawn Numbers ***

450 CLS:PRINT
460 FOR N=1 TO D
470 PRINTTAB(18)D$(N)
480 FOR N2=1 TO 400:NEXT N2
490 NEXT N
500 PRINT
510 PRINTTAB(10)"== Hit any key =="
520 A$=INKEY$:IF A$=""GOTO 520
530 GOTO 330
```

Chapter 16



Candy Store

Games that purport to educate defenseless children are popular among parents, chiefly because of the satisfaction of improving the child's mind without actually having to participate. Such games usually have inviting names, like Bunnyland and so forth, which lull the unsuspecting child into a receptive mood before springing the educational aspect.

Some of these games have little real educational value, but only dredge up some learning aspect in order to make the game more palatable to the parents. You all have heard the arguments about arcade games improving hand-eye coordination and motor skills, while fanning the fires of healthy youthful competition.

Candy Store's pretense at education is to teach young people about free enterprise. We all know that it won't, but it was worth a try, anyway.

The object of the game is to operate a candy store profitably for one five-day week. Small, medium, and large bars may be made, in any proportion that the owner's funds and judgment allow. Small bars cost two cents each to make, medium cost 12 cents, and large cost 22 cents.

The candy bars of each size may be repriced each day at any price the operator wishes. However, the computer randomly selects an optimum price for each size at the beginning of the week.

If a day's selling price for a size is lower than the optimum for the week, the store will quickly sell out its entire day's stock. The owner could have made and sold more candy bars in that size, or charged a higher price and made additional profits.

However, if the selling price chosen is higher than the optimum, consumers will not be willing to buy all the candy bars available. Some will be left to go stale. As the margin between the optimum and selling price increases, so does the proportion of stale candy bars.

The goal of the proprietor is to deduce the optimum selling price for each size early enough in the week to sell the maximum number of candy bars at the highest price the market will bear. The owner

A$	Used in INKEY$ loop
CASH	Player's funds
DU	Dummy variable for RND(1)
DY$(n)	Name of day of week
F$	Print using format
LARGE	Money spent to make large bars
LBARS	Number large bars made
LP	Price of large bars
MBARS	Number of medium bars made
MEDIUM	Money spent to make medium bars
MP	Price of medium bars
N	Loop counter
O1-O3	Optimum price of small, medium, and large
P1-P3	Profit made on small, medium, and large
S1-S3	Number sold of small, medium, and large
SBARS	Number of small bars made
SMALL	Money spent on small bars
SP	Selling price

Fig. 16-1. Variables used in Candy Store.

may choose to let some candy bars go stale on purpose, if the higher price will bring in more profits than a sell-out at the optimum.

In the real world, it's a common practice to raise prices with the knowledge that some customers will be lost but the profits will be higher on fewer sales.

On second thought, perhaps this game does have educational value. The optimum values, O1-03, are first selected in lines 380-400. O1 will range from 1 to 11 cents; O2 will range from 17 to 27 cents; while O3 may be as little as 32 cents, and as much as 43 cents. The player can only deduce these optimum prices by seeing how much candy sells at which price levels.

A for-next loop at line 410 repeats through five days' sales. It displays the day of the week and cash remaining first, and asks how many small bars to make at two cents each. This is repeated for medium and large bars, and then the operator is asked to price these items. You can price them lower than your cost, if you wish, but profits are likely to suffer.

Next, the day's results are printed to the screen. If the selling price for each size is less than or equal to the optimum, the stock is sold out. Prices that are too high leave some to go stale, the proportion increasing with the disparity.

After five days, the optimum prices are displayed, so the player can kick him/herself. Learning about free enterprise is always painful, however.

A list of the variables used in the program is shown in Fig. 16-1.

Program Listing

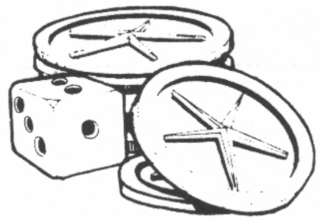
```
 10 ' ***************
 20 ' *             *
 30 ' * Candy Store *
 40 ' *             *
 50 ' ***************

 55 ' *** Set Random Start Point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N

 85 ' ***  Read Days of Week ***

 90 FOR N=1 TO 5
100 READ DY$(N)
110 NEXT N
120 DATA Monday,Tuesday,Wednesday
130 DATA Thursday,Friday

135 ' *** Instructions ***

140 CLS:PRINT:PRINT
150 PRINTTAB(12)"Instructions?"
160 PRINTTAB(16)"Y/N"
170 A$=INKEY$:IF A$=""GOTO 170
180 IF A$="Y" OR A$="y" GOTO 190 ELSE GOTO 360
190 CLS:PRINT
200 PRINTTAB(2)"You have five days to operate"
210 PRINTTAB(2)"your own candy store.  The"
220 PRINTTAB(2)"computer will select a secret"
230 PRINTTAB(2)"optimum price for small, medium"
240 PRINTTAB(2)"and large candy bars."
250 PRINT
260 PRINTTAB(10)"== Hit any key ==";
270 A$=INKEY$:IF A$=""GOTO 270
280 CLS:PRINT
290 PRINTTAB(2)"You set your prices and decide"
300 PRINTTAB(2)"on product mix.  The sooner
```

```
310 PRINTTAB(2)"you guess the optimum prices"
320 PRINTTAB(2)"the more you may make."
330 PRINT
340 PRINTTAB(10)"== Hit any key =="
350 A$=INKEY$:IF A$=""GOTO 350
360 F$="$$####.##"
370 CASH=50

375 ' *** Choose Optimum ***

380 O1=INT(RND(1)*12)
390 O2=INT(RND(1)*12)+16
400 O3=INT(RND(1)*12)+31

405 ' *** How many to make? ***

410 FOR N=1 TO 5
420 CLS:PRINT
430 PRINTTAB(2)"Today is ";DY$(N);"."
440 PRINTTAB(2)"You have $";CASH
450 PRINTTAB(2)"How much for small bars?"
460 PRINTTAB(2)"They cost 2 cents each to make."
470 PRINTTAB(2)"";
480 INPUT SMALL$
490 SMALL=VAL(SMALL$)
500 IF SMALL>CASH GOTO 480
510 SBARS=INT(SMALL/.02)
520 CASH=CASH-SBARS*.02
530 CLS:PRINT
540 PRINTTAB(2)"You have $";CASH
550 PRINTTAB(2)"How much for medium bars?"
560 PRINTTAB(2)"They cost 12 cents each to make."
570 PRINTTAB(2)"";
580 INPUT MEDIUM$
590 MEDIUM=VAL(MEDIUM$)
600 IF MEDIUM>CASH GOTO 580
610 MBARS=INT(MEDIUM/.12)
620 CASH=CASH-MBARS*.12
630 CLS:PRINT
640 PRINTTAB(2)"You have $";CASH
650 PRINTTAB(2)"How much for large bars?"
```

```
660 PRINTTAB(2)"They cost 22 cents each to make."
670 PRINTTAB(2)"";
680 INPUT LARGE$
690 LARGE=VAL(LARGE$)
700 IF LARGE>CASH GOTO 680
710 LBARS=INT(LARGE/.22)
720 CASH=CASH-LBARS*.22

725 ' *** Set Prices ***

730 CLS:PRINT
740 PRINTTAB(2)"What price for small today?"
750 PRINTTAB(2)"";
760 INPUT SP$
770 SP=VAL(SP$)
780 PRINTTAB(2)"What price for medium today?"
790 PRINTTAB(2)"";
800 INPUT MP$
810 MP=VAL(MP$)
820 PRINTTAB(2)"what price for large today?"
830 PRINTTAB(2)"";
840 INPUT LP$
850 LP=VAL(LP$)

855 ' *** Print Day's Results ***

860 CLS
870 IF SP<=O1 THEN S1=SB:GOTO 900
880 S1=SB*O1/SP
890 GOTO 920
900 PRINTTAB(1)"You sold out small bars at";SP;"."
910 GOTO 940
920 PRINTTAB(1);
930 PRINT INT(SB-S1)"bars were unsold at";SP;"."
940 PRINTTAB(1)"You made ";
950 P1=S1*SP/100
960 PRINT USING F$;P1
970 IF MP<=O2 THEN S2=MB:GOTO 1000
980 S2=MB*O2/MP
990 GOTO 1020
1000 PRINTTAB(1)"You sold out medium bars at";MP;"."
```

```
1010 GOTO 1040
1020 PRINTTAB(1);
1030 PRINT INT(MB-S2)"bars were unsold at";MP;"."
1040 PRINTTAB(1)"You made ";
1050 P2=S2*MP/100
1060 PRINTUSING F$;P2
1070 IF LP<=O3 THEN S3=LB:GOTO 1100
1080 S3=LB*O3/LP
1090 GOTO 1120
1100 PRINTTAB(1)"You sold out large bars at";LP;"."
1110 GOTO 1140
1120 PRINTTAB(1);
1130 PRINT INT(LB-S3)"bars were unsold at";LP;"."
1140 PRINTTAB(1)"You made ";
1150 P3=S3*LP/100
1160 PRINTUSING F$;P3
1170 CASH=CASH+P1+P2+P3
1180 PRINTTAB(10)"== Hit any key =="
1190 A$=INKEY$:IF A$="" GOTO 1190
1200 NEXT N

1205 ' *** Print Week's Results ***

1210 CLS:PRINT
1220 PRINTTAB(1)"Optimum small price was";O1
1230 PRINTTAB(1)"Optimum medium price was";O2
1240 PRINTTAB(1)"Optimum large price was";O3
1250 PRINT
1260 PRINTTAB(1)"You finish with ";
1270 PRINT USING F$;CASH
1280 PRINT"== Hit any key =="
1290 A$=INKEY$:IF A$=""GOTO 1290
1300 CLS:PRINT:PRINT
1310 PRINTTAB(12)"Play again?"
1320 A$=INKEY$:IF A$=""GOTO 1320
1330 IF A$="Y" OR A$="y" THEN RUN
1340 CLS
```

Chapter 17



Craps

A dice game like this is, I believe, the only game in this book that has been mentioned in the Bible. That review was somewhat derogatory, however. Nevertheless, Craps remains a time-honored Las Vegas and Atlantic City pastime that is relatively simple to translate into Model 100 BASIC.

The game is easy enough to program that my oldest son selected it as an exercise for his science fair project last year. He had to have something simple enough to be understood by the judges at the fair. This version is painless to type in and works pretty well.

For those of you unfamiliar with the game, (sure) the object is to roll a winning throw of the dice. In this version, the player bets first. On the first roll, if the player comes up with a 7 or 11, he or she wins the bet automatically. Rolling a 2, 3, or 12 on the first roll is an automatic loss of the bet.

Otherwise, the first roll becomes the *point*. The idea then is to roll one's point before rolling to another seven. The player gets to keep rolling until one or the other turns up. An easy point like six will be more likely to be made than one like 10. To make things a little frustrating, no side bets are allowed. So, if you bet small, and then roll an easy point, it's your tough luck.

The dice are rolled in a routine similar to that used in Avarice. After the first roll, the program checks to see if PT is a 2, 3, or 12 (a loss), or a 7 or 11 (a win). If it is any of these, the round is over, and the program branches to the subroutines at lines 380 and 510, which add or subtract the BET from CASH as appropriate.

Bankruptcy causes a fast trip to line 630, where the bad news is revealed and an invitation to play again extended. You will find a list of the variables used in Fig. 17-1.

Sorry I could not provide a handy wall to toss the dice up against, but then, Craps for your Model 100 would not be quite so portable.

A$	Used in INKEY$ loop
BET	Player's bet
CASH	Amount of money player has remaining
D1	Die #1
D2	Die #2
DU	Dummy variable for RND(1)
N	Loop counter
PT	Player's point
ROLL	Total of most recent roll of dice

Fig. 17-1. Variables used in Craps.

Program Listing

```
 10 ' ***********
 20 ' *         *
 30 ' *  Craps  *
 40 ' *         *
 50 ' ***********

 55 ' *** Set Random Start Point ***

 60 CASH=200
 70 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 80 DU=RND(1)
 90 NEXT N
100 GOTO150

105 ' *** Roll Dice ***

110 D1=INT(RND(1)*6)+1
120 D2=INT(RND(1)*6)+1
130 ROLL=D1+D2
140 RETURN
150 GOSUB110
160 IF CASH<1 GOTO630
170 PT=ROLL

175 ' *** Start Round ***

180 CLS:PRINT
```

```
190 PRINTTAB(6)"Cash remaining $";CASH:PRINT
200 PRINTTAB(2)"How much would you like to bet?"
210 PRINTTAB(2)"";
220 INPUT BET
230 IF BET>CASH GOTO180
240 CLS:PRINT:PRINT
250 IF PT=2 OR PT=3 OR PT=12 GOTO380
260 IF PT=7 OR PT=11 GOTO510

265 ' *** Your Point ***

270 CLS:PRINT
280 PRINTTAB(2)"Your point is ";PT
290 PRINT
300 PRINTTAB(2)"Rolling dice..."
310 PRINT
320 GOSUB110
330 PRINTTAB(2)"You rolled a ";ROLL
340 FORN=1 TO 500:NEXT N
350 IF ROLL=PT THEN GOTO550
360 IF ROLL=7 GOTO420
370 GOTO270

375 ' *** You lose ***

380 CLS:PRINT:PRINT
390 PRINTTAB(2)"First roll was";PT
400 PRINT
410 GOTO450
420 CLS:PRINT:PRINT
430 PRINTTAB(9)"Roll was "ROLL
440 PRINT
450 PRINTTAB(12)"You lose!!!"
460 CASH=CASH-BET
470 PRINT
480 PRINTTAB(8)"== Hit any key =="
490 A$=INKEY$:IF A$=""GOTO490
500 GOTO150

505 ' *** You win ***

510 CLS:PRINT:PRINT
```

```
520 PRINTTAB(2)"First roll was"PT
530 PRINT
540 GOTO580
550 CLS:PRINT:PRINT
560 PRINTTAB(9)"Roll was "ROLL
570 PRINT
580 PRINTTAB(12)"You win!!"
590 CASH=CASH+BET
600 PRINT:PRINTTAB(8)"== Hit any key =="
610 A$=INKEY$:IF A$=""GOTO610
620 GOTO150

625 ' *** You are bankrupt ***

630 CLS:PRINT:PRINT
640 PRINTTAB(2)"You are bankrupt!!"
650 PRINT
660 PRINTTAB(8)"Play again?"
670 PRINT
680 PRINTTAB(12)"(Y/N)"
690 A$=INKEY$:IF A$=""GOTO690
700 IF A$="Y"OR A$="y" THEN RUN
```

Chapter 18

T? ?E ?R N?T ??

Phrase Guess

Hangman is for sissies. Highly intelligent Model 100 owners would never stoop to guessing a mere word. Instead, we tackle whole phrases at one grasp and challenge another player to stump us in the process. Phrase Guess is a guess-the-letter game for two players, who each enter a phrase, like "run for the money", or "Om mani padme hum" for the other to guess.

Each time a player guesses a letter in the phrase correctly, another turn is granted. A letter may appear more than once in the phrase, but only the first occurrence is displayed for each guess. Therefore, if T appears three times, the player must guess it three times. Letters guessed already appear along the bottom of the screen. An incorrect guess gives the other player a chance to guess.

Phrase Guess has some really nifty features, such as automatic conversion to uppercase, and a limited type of screen formatting. It also makes extensive use of the player's actual names. This is done to avoid confusion as to whose turn it is and who is entering a word to confuse which other player.

The names are stored in a string array, N$(n), in line 230. Then, another for-next loop from 1 to 2 asks player N$(N) to enter a phrase for player N$(OP). Thanks to line 290, if N=1 then OP will equal 2 and vice versa. Because line input is used in line 310, the phrase can contain a comma—but should not. Otherwise, the opponent will never guess that letter.

The phrases entered are first changed to all uppercase letters. A for-next loop from 1 to the length of the phrase (E$(OP)) is repeated. If a character's ASCII code is greater than 96 (indicating a lowercase letter), it is reduced by 32 to change it to uppercase. Then the program constructs the string P$(n), which has hyphens in the positions of letters in the phrase and spaces where spaces actually occur. Thus, the player can tell how many letters there are in each word.

The actual "---" representation of the phrase is

A$	Used in INKEY$ loop
E$(n)	Array storing phrases entered by players
G$(n)	Array storing letters guessed by players
N	Loop counter
N2	Loop counter
OP	Opponent
P$(n)	Array storing "---" representation
PLAYER	Current player
Q	Amount to tab to center phrase
R$	Character in middle of E$(n)
T	Position of first occurrence of letter
Y$	Character in middle of E$(n)

Fig. 18-1. Variables used in Phrase Guess.

displayed, centered in the screen. It might be interesting to look at how the centering is accomplished. The program finds the length of the phrase using LEN and then subtracts that from 40, the width of the screen, to determine how many blanks are left over. By dividing this number by two, and tabbing that far over, an equal number of blanks are presented before and after the phrase. It is centered.

A player guesses a letter at the beginning of each round. If lowercase letters are guessed, they are changed to uppercase in line 550. The guessed letter is added to the string of letters guessed by that player, G$(PLAYER). Then, a check is made using INSTR to see if A$ is contained within the phrase E$(PLAYER). Because INSTR is used, the variable T takes the value of the position of the first occurrence of that letter.

If T=0, then the guess is incorrect, and the turn is over. If PLAYER=1, then PLAYER=2, and vice versa, due to an operation at line 660. The keyword SWAP would be nice in Model 100 BASIC.

If, however, T< >0, the appropriate hyphen in P$(PLAYER) is replaced by the letter, and the letter in E$(PLAYER) is replaced by a hyphen. The last is done so that if that letter is guessed again, the guess will be incorrect unless the letter appears at another location in the phrase.

When all the hyphens are gone from P$ (PLAYER), in line 460, the player has guessed all the letters in the phrase, and has won. If the player's partner is not a sore loser, another round can be played and two more difficult phrases entered.

The variables used in the program are shown in Fig. 18-1.

Program Listing

```
10 ' ****************
20 ' *              *
30 ' * Phrase Guess *
40 ' *              *
50 ' ****************
```

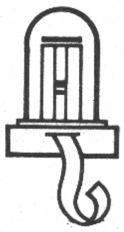
```
 55 ' *** Instructions ***
 60 CLS:PRINT:PRINT
 70 PRINTTAB(12)"Instructions?"
 80 PRINT:PRINTTAB(16)"(Y/N)"
 90 A$=INKEY$:IF A$=""GOTO 90
100 IF A$="Y" OR A$="y" GOTO 110 ELSE  GOTO 190
110 CLS:PRINTTAB(6)"Two players attempt to guess"
120 PRINTTAB(6)"a phrase entered by the other."
130 PRINTTAB(6)"If you guess a correct letter,"
140 PRINTTAB(6)"you get another turn.  First"
150 PRINTTAB(6)"to get whole phrase wins."
160 PRINT
170 PRINTTAB(10)"== Hit any key =="
180 A$=INKEY$:IF A$=""GOTO 180

185 ' *** Enter Player Names ***

190 FOR N=1 TO 2
200 CLS:PRINT:PRINT
210 PRINTTAB(2)"Enter name Player #";N
220 PRINTTAB(2)"";
230 INPUT N$(N)
240 NEXT N

245 ' *** Enter Phrases ***

250 FOR N=1 TO 2
260 CLS:PRINT:PRINT
270 PRINTTAB(2)N$(N);", enter a phrase"
280 PRINTTAB(2)"for ";
290 IF N=1 THEN OP=2 ELSE  OP=1
300 PRINT N$(OP);" to guess."
310 LINEINPUT E$(OP)

315 ' *** Change Phrase to All Uppercase ***

320 FOR N2=1 TO LEN(E$(OP))
330 Y$=MID$(E$(OP),N2,1)
340 IF Y$="." GOTO 360
350 IF ASC(Y$)>96 THEN MID$(E$(OP),N2,1)=CHR$(ASC(Y$)-32)
360 NEXT N2
```

```
370 NEXT N

375 ' *** Construct "---" ***

380 FOR N=1 TO 2
390 FOR N2=1 TO LEN(E$(N))
400 R$=MID$(E$(N),N2,1)
410 IF R$=CHR$(32) THEN P$(N)=P$(N)+CHR$(32) ELSE
    P$(N)=P$(N)+"-"
420 NEXT N2
430 NEXT N
440 PLAYER=1

445 ' *** Start Round ***

450 CLS:PRINT
460 IF INSTR(P$(PLAYER),"-")=0 GOTO 700
470 PRINTTAB(6)"It is ";N$(PLAYER);"'s turn."
480 PRINT:PRINT
490 Q=INT(40-LEN(P$(PLAYER)))/2
500 PRINTTAB(Q)P$(PLAYER)
510 PRINT
520 PRINTTAB(4)G$(PLAYER)
530 PRINT@285,"Guess a letter :";
540 A$=INKEY$:IF A$=""GOTO 540

545 ' *** Correct Guess ***

550 IF ASC(A$)>90 THEN A$=CHR$(ASC(A$)-32)
560 G$(PLAYER)=G$(PLAYER)+A$
570 T=INSTR(E$(PLAYER),A$)
580 IF T=0 GOTO 620
590 MID$(P$(PLAYER),T,1)=A$
600 MID$(E$(PLAYER),T,1)="-"
610 GOTO 450

615 ' *** Wrong Guess ***

620 CLS:PRINT
630 PRINTTAB(10)"Wrong guess."
640 PRINT
```

```
650 PRINTTAB(8)"Your turn is over."
660 IF PLAYER=1 THEN PLAYER=2 ELSE  PLAYER=1
670 PRINT:PRINTTAB(8)"== Hit any key =="
680 A$=INKEY$:IF A$="" GOTO 680
690 GOTO 450

695 ' *** Game Over ***

700 CLS:PRINT
710 PRINTTAB(8)N$(PLAYER);" won!"
720 PRINT
730 PRINTTAB(12)"Play again?"
740 A$=INKEY$:IF A$=""GOTO 740
750 IF A$="Y" OR A$="y" THEN RUN
760 CLS
```

Chapter 19

Stock Market



"Buy low—sell high." It sounds so simple, yet so many investors get it exactly backwards. If we would only pay attention to the stock market, many more of us would make a killing. However, if you are lacking one or more of the essentials, such as funds to invest, you can play this version of Stock Market on your very own Model 100 computer.

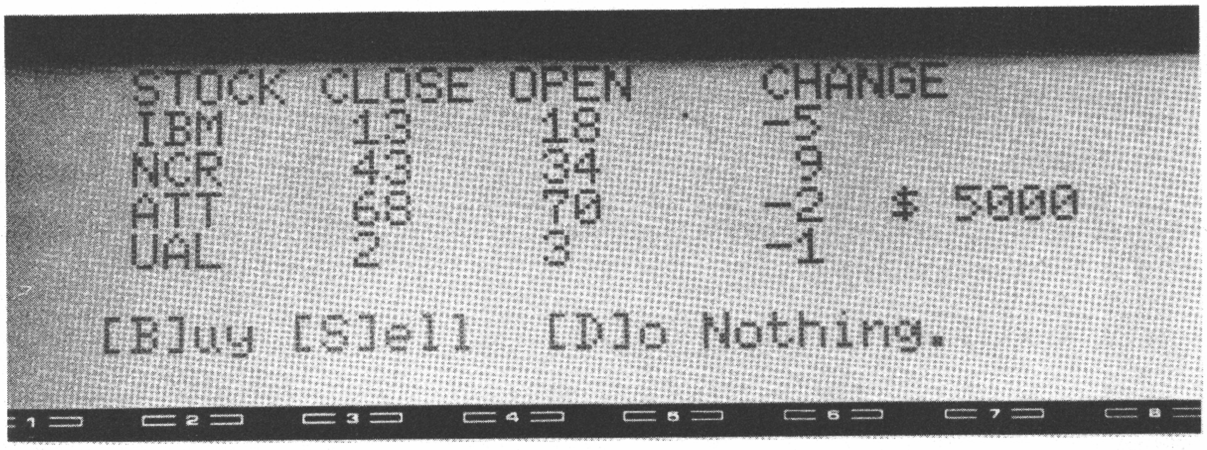
The object is to amass a huge quantity of money, enough, preferably, to overflow the Model 100 screen and ruin the careful formatting I have done. This will require a great deal of money. Unlike the real stock market, you have just four stocks to choose from: IBM, NCR, ATT, and UAL, so the choices should not be overwhelmingly difficult.

Each turn, the values of the stocks go up or down depending largely on whether or not you own any of that stock or not—just kidding! You may buy shares, sell shares, or do nothing and sit on your holdings. That may be all you have left to sit on if your decisions are not shrewd.

The program loops through the number of rounds entered by the player in line 240. Then, the initial price of the four stocks is set, along with a first change for you trend-watchers. The price of each stock, PR(n), is set to a random number less than 100 in line 270. Then, a change between one and 19 dollars is selected. This change may be up or down, depending on the value of a third random number, F. Then, the opening price, which is the current price, PR(n), plus the change, CH (and CH can be a negative number, remember). If the opening price is less than one, it is set equal to one. You don't want any stocks selling for less than nothing, in order to avoid a run on that issue.

The turn starts at line 340, where the for-next loop commences. The stock name, closing price, opening price, and the difference between the two (the change) is displayed as shown in Fig. 19-1. Then, the player can enter B for buy, S for sell, or D for do nothing. Depending on the option entered, control passes to line 510, 750, or 990.

In the first routine, the stock names, current



```
STOCK CLOSE OPEN    CHANGE
IBM    13    18      -5
NCR    43    34       9
ATT    68    70      -2  $ 5000
UAL    2     3       -1

[B]uy [S]ell  [D]o Nothing.
```

Fig. 19-1. Screen display from Stock Market.

price, and number of shares owned (SH(n)) are displayed. The player elects to buy a stock by entering the number which appears next to that stock on the big board. Then, he or she is asked how much will be invested. It is impossible to spend more than you have in this game. Were that real life were similar! The money specified is used to buy the stock. No fractional shares are purchased. If money is left over, it is returned to the player's CASH kitty.

The player can buy more or return to the menu to be offered the same three choices again. The price of all the stocks remain the same until D for do nothing is entered. Selling stock is done similarly to buying, except that the player is asked how many shares of stock are to be sold. You may not sell short, that is, sell more shares than you have in this game.

The do nothing option merely triggers a new round of random number generation. Factor F again

A$	Used in INKEY$ loop
A	Value of A$
CASH	Player's money
CH	Change in price
DU	Dummy variable for RND(1)
F	Factor, +1 or −1; shows direction of change
INVEST$	Money invested in a stock
N	Loop counter
OP(n)	Opening price
PR(n)	Current price of stock
ROUND	Round of play
SB	Stock bought
SH(n)	Shares held of a given stock
SOLD	Stock sold
ST$(n)	Name of stock
TURN	Turn of game, loop counter

Fig. 19-2. Variables used in Stock Market.

determines whether the price will go up or down, while CH is given a random value from 1 to 19. Unlike the real stock market, prices can go up or down only in integers, and jumps are limited. A list of the variables used in the program is shown in Fig. 19-2.

Program Listing

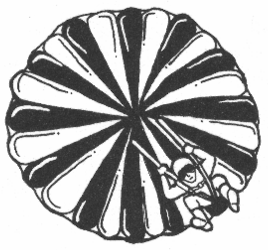
```
 10 ' ****************
 20 ' *              *
 30 ' * Stock Market *
 40 ' *              *
 50 ' ****************

 55 ' *** Set Random Start Point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N
 90 CASH=5000
100 DATA IBM,NCR,ATT,UAL

105 ' *** Instructions ***

110 CLS:PRINT:PRINT
120 PRINTTAB(12)"Instructions?"
130 PRINTTAB(16)"Y/N"
140 A$=INKEY$:IF A$="" GOTO 140
150 IF A$="Y" OR A$="y" GOTO 160 ELSE GOTO 210
160 CLS:PRINT
170 PRINTTAB(9)"Buy low, sell high."
180 PRINT:PRINT
190 PRINTTAB(9)"== Hit any key =="
200 A$=INKEY$:IF A$=""GOTO 200

205 ' *** How many rounds? ***

210 CLS:PRINT
220 PRINTTAB(4)"How many rounds to play"
230 PRINTTAB(8)"";:INPUT ROUND$
240 ROUND=VAL(ROUND$)

245 ' *** Set Initial Prices ***

250 FOR N=1 TO 4
```

```
260 READ ST$(N)
270 PR(N)=INT(RND(1)*100)
280 CH=INT(RND(1)*20)
290 F=INT(RND(1)*2)+1:IF F>=2 THEN F=-1
300 CH=CH*F
310 OP(N)=PR(N)+CH
320 IF OP(N)<1 THEN OP(N)=1
330 NEXT N

335 ' *** Start Turn ***

340 FOR TURN=1 TO ROUND
350 CLS:PRINT
360 PRINTTAB(5)"STOCK";TAB(11)"CLOSE";TAB(17)
    "OPEN";TAB(25);"CHANGE"
370 FOR N=1 TO 4
380 PRINTTAB(5) ST$(N);
390 PRINTTAB(11)PR(N);TAB(17);OP(N);TAB(25);
    PR(N)-OP(N)
400 NEXT N
410 PRINT
420 PRINTTAB(4)"[B]uy [S]ell  [D]o Nothing."
430 PRINT@149,"$";CASH
440 A$=INKEY$:IF A$=""GOTO 440
450 IF A$="B" OR A$="b" GOTO 510
460 IF A$="S" OR A$="s" GOTO 750
470 IF A$="D" OR A$="d" GOTO 990
480 GOTO 440
490 NEXT TURN
500 GOTO 1080

505 ' *** Buy Stock ***

510 CLS
520 PRINTTAB(2)"Stock";TAB(12);"Price";TAB(22);"Shares owned"
530 FOR N=1 TO 4
540 PRINTTAB(2)N;".)";ST$(N);TAB(14);PR(N);TAB(24)SH(N)
550 NEXT N
560 PRINT
570 PRINTTAB(6)"Enter number stock wanted."
580 PRINT@149,"$";CASH
```

```
590 A$=INKEY$:IF A$=""GOTO 590
600 A=VAL(A$)
610 IF A<1 OR A>4 GOTO 590
620 PRINT@288,"How much to invest";
630 INPUT INVEST$
640 INVEST=VAL(INVEST$)
650 IF INVEST>CASH GOTO 620
660 SB=INT(INVEST/PR(A))
670 SH(A)=SH(A)+SB
680 CASH=CASH-SB*PR(A)
690 CLS:PRINT
700 PRINTTAB(13)"Buy more?"
710 PRINT
720 PRINTTAB(16)"Y/N"
730 A$=INKEY$:IF A$=""GOTO 730
740 IF A$="Y" OR A$="y" GOTO 510 ELSE GOTO 350

745 ' *** Sell Stock ***

750 CLS
760 PRINTTAB(2)"Stock";TAB(12);"Price";TAB(22);"Shares owned"
770 FOR N=1 TO 4
780 PRINTTAB(2)N;".)";ST$(N);TAB(14);PR(N);TAB(24)SH(N)
790 NEXT N
800 PRINT
810 PRINTTAB(6)"Enter number stock to sell."
820 PRINT@149,"$";CASH
830 A$=INKEY$:IF A$=""GOTO 830
840 A=VAL(A$)
850 IF A<1 OR A>4 GOTO 830
860 PRINT@288,"How many shares to sell";
870 INPUT INVEST$
880 INVEST=VAL(INVEST$)
890 IF INVEST>SH(A) GOTO 860
900 SOLD=INVEST*PR(A)
910 SH(A)=SH(A)-INVEST
920 CASH=CASH+SOLD
930 CLS:PRINT
940 PRINTTAB(13)"Sell more?"
950 PRINT
960 PRINTTAB(16)"Y/N"
```

```
 970 A$=INKEY$:IF A$=""GOTO 970
 980 IF A$="Y" OR A$="y" GOTO 750 ELSE GOTO 350

 985 ' *** Change Values ***

 990 FOR N=1 TO 4
1000 OP(N)=PR(N)
1010 F=INT(RND(1)*2)
1020 IF F=0 THEN F=-1
1030 CH=INT(RND(1)*20)
1040 PR(N)=PR(N)+CH*F
1050 IF PR(N)<1 THEN PR(N)=1
1060 NEXT N
1070 GOTO 490

1075 ' *** Game Over ***

1080 CLS:PRINT
1090 PRINTTAB(12)"Game Over!"
1100 PRINT
1110 PRINTTAB(6)"You finished with $";CASH
1120 PRINT
1130 PRINTTAB(12)"Play again?"
1140 A$=INKEY$:IF A$=""GOTO 1140
1150 IF A$="Y" OR A$="y" THEN RUN
1160 CLS
```

Chapter 20



Skydiver

Computer games give us the chance to try risky maneuvers without really taking any chances. Skydiver closely duplicates the thrills of actual skydiving with all the excitement of wind in your face, zero gravity, and the tug of a chute opening up. If you believe that, I know where you can get a good deal on a used bridge.

Actually, Skydiver is an amusing variation on the bomber games. Instead of dropping bombs on a target, the plane drops a parachutist, who will attempt to land on a landing pad. The pad can be from one to four blocks wide, depending on the difficult level set by the player. Ten points are gained for each successful landing. A miss results in one point being subtracted from your score.

In line 250, the player enters the difficulty desired, ranging from one to four. Any number larger than four is rejected. Making a landing pad 40 spaces wide on the bottom of the screen would be cheating. A random location, T, is selected, and a for-next loop from one to the number selected as difficulty prints a graphics block, CHR$(239), at T+N each time through the loop.

A > (greater than) symbol is used to represent the airplane (maybe, a jet), because the plane in the Model 100's character set travels only vertically up the screen. The graphics symbol for spades looked a lot like a parachutist, so I used that, CHR$(159) in this program.

One note about the use of graphics symbols within programs. You can, of course, type PRINT"*", or whatever, and the Model 100 will print that character to the screen. But, once you depart from the standard alphanumerics, there will be a problem listing that symbol on your printer or sending the file to another user. Their Model 100 will understand what you mean, but another computer or computer system, such as CompuServe, will be mightily confused.

Instead, define some variable, such as PLANE$ or CHUTE$ with the appropriate CHR$ code. Your printer will love you for it. In addition,

A$	Used in INKEY$ loop
DF	Difficulty level; size of landing pad
DU	Dummy variable for RND(1)
N	Loop counter
PARA$	Graphics character for parachute
PLANE$	Graphics character for plane
SC	Player's score
T	Random location of landing pad

Fig. 20-1. Variables used in Skydiver.

should you decide to switch to another graphics character, the only change that need be made will be in the definition line.

Once the screen has been cleared, and the player's current score printed, an INKEY$ loop commences. At location A, which initially has a value of 2, the program prints PLANE$, and puts a space, CHR$(32) at A−1, which is the previous location of the plane. Then A is incremented, and a brief for-next loop provides a delay. This process repeats until a key is pressed. At that point, the image of the plane is erased from the screen, in line 430, and the parachutist is printed at A in its place.

When A was increased by one each time, the movement of the plane was horizontal. Now you increase A by 41, so that the parachutist moves down one row and over one space each time, producing a diagonal fall. A longer loop in line 450 makes this descent appear to be slow and graceful. When the chute's patch crosses the landing pad (PEEK(A−512)=239), the player is deemed to have landed safely. Ten points are awarded in a routine beginning at line 510.

Reaching the bottom of the screen, in line 480, without hitting the pad leads to another module, at line 590, which subtracts a point and provides a brief display of sound and exploding human. The game can be repeated until the player accumulates a new high in points, decides to increase the difficulty, or gets the urge to try skydiving for real. This would probably not be a good game to play while aboard an airplane, especially around nervous first-time flyers.

A list of variables is found in Fig. 20-1.

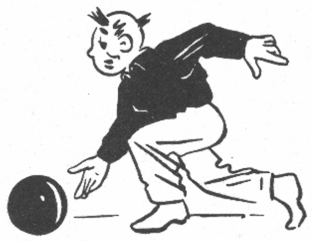
Program Listing

```
10 ' *************
20 ' *           *
30 ' * Skydiver *
40 ' *           *
50 ' *************

55 ' ** Set Random Start Point ***

60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
70 DU=RND(1)
80 NEXT N

85 ' *** Instructions ***

90 CLS:PRINT:PRINT
```

```
100 PRINTTAB(12)"Instructions?"
110 PRINT:PRINTTAB(16)"Y/N"
120 A$=INKEY$:IF A$=""GOTO120
130 IF A$="Y" OR A$="y" GOTO140ELSE GOTO220
140 CLS:PRINT
150 PRINTTAB(6)"Difficulty level sets size"
160 PRINTTAB(6)"of the landing pad.  Hit any"
170 PRINTTAB(6)"key to release chute.  You get"
180 PRINTTAB(6)"10 points for a safe landing."
190 PRINTTAB(6)"You lose one for a crash."
200 PRINT:PRINTTAB(9)"== Hit any key =="
210 A$=INKEY$:IF A$="" GOTO210

215 ' *** Set Size of Pad ***

220 CLS:PRINT
230 PRINTTAB(5)"Enter difficulty level:"
240 PRINT:PRINTTAB(6)"[1] Hard to [4] Easy"
250 A$=INKEY$:IF A$="" GOTO250
260 DF=VAL(A$)
270 IF DF>4 GOTO250
280 T=INT(RND(1)*30)+285

285 ' *** Start Round ***

290 CLS
300 A=2
310 FOR N=1 TO DF
320 PRINT@T+N,CHR$(239);
330 NEXT N
340 PARA$=CHR$(159)
350 PLANE$=CHR$(62)
360 PRINT@28,"Score :";SC;
370 A$=INKEY$
380 PRINT@A,PLANE$;
390 PRINT@A-1,CHR$(32);
400 A=A+1
410 FOR N=1 TO 100:NEXT N
420 IF A$=""GOTO360
430 PRINT@A-1,CHR$(32);
440 PRINT@A,PARA$;
```

```
450 FOR N=1 TO 500:NEXT N
460 PRINT@A,CHR$(32);
470 A=A+41
480 IF A>320 THEN A=A-41:GOTO590
490 IF PEEK(A-512)=239 GOTO510
500 GOTO440

505 ' *** Landed on Pad ***

510 CLS:PRINT
520 PRINTTAB(12)"Nice landing!"
530 PRINTTAB(12)"You get ten points."
540 PRINT
550 PRINTTAB(12)"== Hit any key =="
560 A$=INKEY$:IF A$=""GOTO560
570 SC=SC+10
580 GOTO280

585 ' *** Missed Pad ***

590 FOR N=1 TO 50
600 PRINT@A,CHR$(190);
610 GOSUB730
620 PRINT@A,CHR$(199);
630 NEXT N
640 CLS:PRINT
650 PRINTTAB(12)"You crashed!"
660 PRINT
670 PRINTTAB(12)"You lose a point."
680 PRINT
690 PRINTTAB(12)"== Hit any key =="
700 SC=SC-1
710 A$=INKEY$:IF A$=""GOTO710
720 GOTO280

725 ' *** Sound ***

730 SOUND 7000,1:RETURN
```

Chapter 21



Bowling

Bowling combines most of the techniques used so far, from peeking the screen to sound. It takes advantage of the aspect ratio of the Model 100 screen, which is much wider than it is deep.

Unlike a real bowling alley, Bowling has no gutters. The pins make beep sounds when they fall, rather than loud noises, and, in this game, a perfect game adds up to only 270.

In play, a bowling ball moves up and down at the left side of the screen. The player may set the speed of travel to make the game easy or difficult. When the correct position is reached, any key is pressed to release the ball. Depending on where the ball hits the pins, the player may get a strike, a split, or a spare. Except that no extra balls are provided after the tenth frame, because of the difficulty of displaying more than 10 frames on the Model 100 screen, scorekeeping is like real bowling.

When you roll a spare, your score for that frame is increased by the number of pins you hit on your next ball. A strike gives you a bonus of the pins hit on the next two balls. While this scoring system seems simple enough, it was a devil to program. Originally, I set up a series of flags, which kept track of whether the previous ball resulted in a strike or spare, and then went back and added to the score as needed.

Not necessary! Because a computer works so fast, it was much simpler just to add up the score anew every single frame. The computer can make the additions over and over in no time at all, so the delay is not noticed. Thus, if a strike is thrown in the first frame, the computer will post a 10 and then add the results for the next two balls, both zero at this point, to arrive at a total score for the first frame of 10.

Suppose that four pins are struck with the first ball of the second frame and three with the last ball of the frame. When the computer does its addition, it will come up with 17 for the first frame and a total of 24 for the second: just like in real bowling. The main difference is that the partial scores are posted pending throwing of the bonus balls.

The print @ positions of the ten pins are placed in data lines in lines 310-340. These are read and loaded into an array, C(n), and used to print P$, defined as CHR$(234), in the correct locations to reproduce a pin setup.

Two balls are thrown each frame, unless a strike is thrown on the first. Variable A is used to keep track of the position of the ball, and this is incremented by DELTA, which is initially given a value of 40. Thus, while the INKEY$ loop waits for the player to strike a key, A increases by 40 each time, gradually moving the ball down the screen. When it reaches the bottom, DELTA becomes −40, and the ball starts back up. This repeats, interrupted only by the delay loop at lines 480 and 500, until the player hits a key.

At this point, A begins to be incremented by a value of one, which moves it horizontally across the screen. A check in line 540 spots collisions with pins and sends the program to a sound routine for a nice beep.

This collision detection has nothing to do with the score, however. Some extra pins are knocked down, their number determined partly by chance and partly by the position at which the ball strikes the pins. The screen row is calculated by dividing G, the point at which the ball started moving, by 40 and adding one. Various program lines, beginning at 620, erase some extra pins.

After this, the number of pins knocked down is determined by peeking memory at the print @ locations stored in C(n) minus 512. Any spaces found are absent pins. On the second ball, the pins that were knocked down by the first ball will still be missing. So, the number of pins hit on the second try is determined by subtracting the number of pins hit by the first ball from the total missing.

The pins hit by each ball are stored in a two

A$	Used in INKEY$ loop
A	Position of ball
B(r,c)	Pins struck by each ball thrown
BALL	Loop counter, two balls for each frame
C(n)	Positions of pins
CU	Counter
DELTA	Chance of A, ball position
DL	Delay loop
DU	Dummy variable for RND(1)
F	Row ball struck pins
FRAME	Frame
G	Position of ball as it heads towards the pins
N	Loop counter
P$	Graphic character for pin
SC(n)	Score for each frame, totaled
T	Random pin drop
TR	Total score to this frame
TT	Total pins dropped in a frame

Fig. 21-1. Variables used in Bowling.

dimensional array, B(FRAME,BALL). The numbers stored in this array are used to determine the score for each frame (SC(n)) by addition, using the common bowling scoring algorithm outlined previously. Then, the score is printed to the screen, and the next frame started. A list of the variables used in the program is found in Fig. 21-1.

Unfortunately, because of the width of the Model 100 screen, it is not possible to display more than 10 frames, given the three-digit scores that are easily obtainable. So, I left the extra balls thrown after the tenth frame out of this program.

Program Listing

```
 10 ' ***********
 20 ' *         *
 30 ' * Bowling *
 40 ' *         *
 50 ' ***********

 55 ' *** Set Random Start Point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N

 85 ' *** Instructions ***

 90 CLS:PRINT:PRINT
100 PRINTTAB(10)"Instructions?"
110 PRINT:PRINTTAB(14)"Y/N"
120 A$=INKEY$:IF A$=""GOTO 120
130 IF A$="Y" OR A$="y" THEN GOTO 140 ELSE GOTO 200
140 CLS:PRINT
150 PRINTTAB(6)"Hit any key to release ball"
160 PRINTTAB(6)"Balls thrown per frame appear"
170 PRINTTAB(6)"at left of screen."
180 PRINT:PRINTTAB(12)"== Hit any key =="
190 A$=INKEY$:IF A$=""GOTO 190

195 ' *** Set Speed ***

200 CLS
210 PRINT:PRINT:PRINTTAB(6)"Enter speed desired:":PRINT
220 PRINTTAB(6)"[1] Fast to [9] Slow"
```

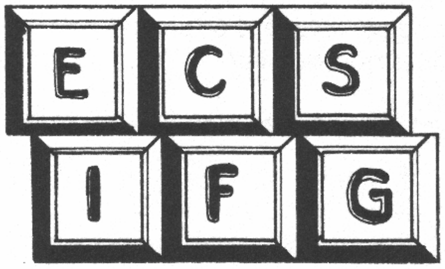
```
230 A$=INKEY$:IF A$=""GOTO 230
240 DL=VAL(A$)*5
250 DIM B(13,2),SC(13)
260 P$=CHR$(234)
270 CLS

275 ' *** Loop through 10 frames ***

280 FOR FRAME=1 TO 10
290 A=1
300 DELTA=40
310 DATA 39,119,199,279
320 DATA 77,157,237
330 DATA 115,195
340 DATA 153
350 FOR N=1 TO 10
360 READ C(N)
370 PRINT@C(N),P$;
380 P(N)=1
390 NEXT N

395 ' *** Throw Ball ***

400 FOR BALL=1 TO 2
410 A=1
420 DELTA=40
430 A$=INKEY$
440 PRINT@A," ";
450 A=A+DELTA
460 IF A>280 THEN DELTA=-40:GOTO 450
470 IF A<1 THEN DELTA=40:GOTO 450
480 FOR N=1 TO DL:NEXT N
490 PRINT@A,"*";
500 FOR N=1 TO DL:NEXT N
510 IF A$="" GOTO 430

515 ' *** Roll Ball toward Pins ***

520 G=A
530 PRINT@A,CHR$(32);
540 IF PEEK(A-511)=234 THEN GOSUB 1320
```

```
550 A=A+1
560 FOR N=1 TO 20:NEXT N
570 PRINT@A,"*";
580 FOR N=1 TO 20:NEXT N
590 IF A MOD 40=0 GOTO 610
600 GOTO 530
610 F=INT(G/40)+1

615 ' *** Knock Down Extra Pins ***

620 ON F GOTO 630,640,670,740,800,850,900
630 GOTO 930
640 PRINT@C(1)," ";
650 GOSUB 1320
660 GOTO 930
670 PRINT@C(1)," ";
680 GOSUB 1320
690 PRINT@C(3)," ";
700 GOSUB 1320
710 PRINT@C(5)," ";
720 GOSUB 1320
730 GOTO 930
740 FOR N=1 TO 3
750 T=INT(RND(1)*10)+1
760 PRINT@C(T)," ";
770 GOSUB 1320
780 NEXT N
790 GOTO 930
800 FOR N=1 TO 10
810 PRINT@C(N)," ";
820 GOSUB 1320
830 NEXT N
840 GOTO 930
850 PRINT@C(4)," ";
860 GOSUB 1320
870 T=INT(RND(1)*2)
880 IF T=1 THEN PRINT@C(2)," ";:GOSUB 1320
890 GOTO 930
900 T=INT(RND(2))
910 IF T=1 THEN PRINT@C(7)," ":GOSUB 1320
```

```
 925 ' *** Count downed pins ***

 930 FOR N=1 TO 10
 940 IF PEEK(C(N)-512)=32 THEN TT=TT+1
 950 NEXT N
 960 B(FRAME,BALL)=TT
 970 IF BALL=2 THEN B(FRAME,2)=B(FRAME,2)-B(FRAME,1)
 980 IF B(FRAME,2)=<0 THEN B(FRAME,2)=0
 990 PRINT@10,"Pins:";B(FRAME,BALL)
1000 TT=0
1010 CU=CU+1
1020 FOR N=1 TO 500:NEXT N
1030 IF BALL=2 GOTO 1060
1040 IF G=161 THEN GOTO 1060
1050 NEXT BALL

1055 ' *** Add up Score ***

1060 FOR N=1 TO FRAME
1070 SC(N)=B(N,1):IF B(N,1)=10 GOTO 1110
1080 SC(N)=SC(N)+B(N,2)
1090 IF SC(N)=10 THEN SC(N)=SC(N)+B(N+1,1)
1100 GOTO 1120
1110 SC(N)=SC(N)+B(N+1,1):IF SC(N)=20 THEN
     SC(N)=SC(N)+B(N+2,1) ELSE SC(N)=SC(N)+B(N+1,2)
1120 NEXT N

1125 ' *** Print Score on Screen ***

1130 CLS
1140 PRINT@280,"";
1150 FOR N=1 TO 10
1160 IF SC(N)=0 GOTO 1210
1170 TR=TR+SC(N)
1180 TR$=MID$(STR$(TR),2)
1190 PRINT TR$;" ";
1210 NEXT N
1220 RESTORE
1230 A=1
1240 TR=0
1250 NEXT FRAME
```

```
1255 ' *** Game Over ***

1260 PRINT@55,"Game Over";
1270 PRINT@135,"Play again?"
1280 A$=INKEY$:IF A$=""GOTO 1280
1290 IF A$="Y"OR A$="y"THEN RUN
1300 CLS
1310 END

1315 ' *** Sound Routine ***

1320 SOUND 7000,1
1330 FOR DE=1 TO 50:NEXT DE
1340 RETURN
```

Chapter 22



Scramble

Memory games are fun to play—but not against a computer, which has an infallible memory. A human opponent to test ones' memory against is more competitive. In Scramble, the Model 100 serves as an unbiased scorekeeper.

What has happened is that the computer has scrambled the letter keys from A-Z. Some letters may be in the right positions. More than likely, pressing an A will yield a Z or some other letter instead. The object is to remember where you see a given letter and to match it up with its correct location. The way to do this is to press a key, see the result on the screen, and then press the key with the letter containing that value. The two values will be switched, leaving at least one of the keys with its correct letter.

For example, say you press the letter A, and a P appears instead. You then press a second key, say Q. If the A is, in fact, stored in Q's position, the A will be restored to proper operation, and the Q will now contain the P.

On first glance, this seems almost trivial. When one presses the A and sees P, one need only press P first next turn, and then A to score a point. However, even for touch typists, this will be more difficult than it seems. And, unless you see which keys your opponent presses, their actions will be of little use to you. Memory plays a part as well. Was it really a P, or an R you saw last time? The first player to make five matches wins.

The keyboard is scrambled using our friendly card-shuffling trick, with the "number of cards" initially set to 26, as there are 26 letters in the alphabet. A random number no larger than the number of letters remaining is selected in line 250, and that letter of the alphabet dropped in that random slot. While it is possible that the random number will simply put a letter back into its own position, this is unlikely, except in the final stages of selection—all the more fun.

The players, addressed by name by the computer, get to press a key in line 410. Variable G is

A$	Used in INKEY$ loop
B	Converts character to lowercase
C$	Letters correctly guessed so far
DU	Dummy variable for RND(1)
F1	Alphabet position of first key pressed
FLAG	Flag showing whether key was first or second
G	ASCII value of key pressed
KB(n)	Array storing scrambled keyboard positions
L	Random number to drop value into
N	Loop counter
NC	Number of keys yet to be distributed
PLAYER$(n)	Name of player
PLAYER	Current player
SC(n)	Player's scores

Fig. 22-1. Variables used in Keyboard Scramble.

given the ASCII value of that key, and if it is a lowercase letter, it is converted to uppercase. All other keyboard input is ignored. The first key pressed is compared to the second in line 510 to see if the letter stored is the desired one. If so, the two are swapped, and the player is given a point. When the score, SC(PLAYER), exceeds five, the program branches to the winning routine, for appropriate laudatory praise. A list of the variables used in the program is shown in Fig. 22-1.

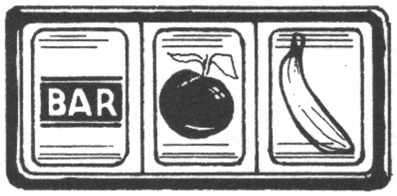
Program Listing

```
 10 ' **************
 20 ' *            *
 30 ' *  Scramble  *
 40 ' *            *
 50 ' **************

 55 ' *** Instructions ***

 60 CLEAR 1000
 70 CLS
 80 PRINT:PRINT
 90 PRINTTAB(12)"Instructions?"
100 PRINT:PRINTTAB(16)"(Y/N)
110 A$=INKEY$:IF A$=""GOTO 110
```

```
120 IF A$="Y" OR A$="y" GOTO 130 ELSE GOTO 180
130 CLS:PRINT
140 PRINTTAB(2)"Press any key A-Z, then try to"
150 PRINTTAB(2)"find the key which has its correct"
160 PRINTTAB(2)"letter.  First player to make five"
170 PRINTTAB(2)"matches wins the game."
180 FLAG=1
190 DIM KBD(26),DU(26)
200 NC=26

205 ' *** Set Random Start Point ***

210 FOR N=1 TO 26
220 DU(N)=N+64
230 NEXT N

235 ' *** Scramble Keyboard ***

240 FOR N=1 TO 26
250 L=INT(RND(1)*NC)+1
260 KBD(N)=DU(L)
270 IF L=NC GOTO 290
280 DU(L)=DU(NC)
290 NC=NC-1
300 NEXT N
310 PRINT:PRINTTAB(12)"==Hit any key =="
320 A$=INKEY$:IF A$="" GOTO 320

325 ' *** Enter Player Names ***

330 FOR N=1 to 2
340 CLS:PRINT:PRINT
350 PRINTTAB(2)"Enter name Player #";N
360 INPUT PLAYER$(N)
370 NEXT N

375 ' *** Start New Turn ***

380 IF PLAYER=1 THEN PLAYER=2 ELSE PLAYER=1
390 CLS:PRINT:PRINTTAB(4)"It is ";PLAYER$(PLAYER);"'s turn:"
400 PRINTTAB(4) PLAYER$(1);":";SC(1);PLAYER$(2);":";SC(2)
```

```
410 A$=INKEY$:IF A$="" GOTO 410
420 G=ASC(A$)
430 IF G<65 GOTO 410
440 IF G>90 AND G<97 GOTO 410
450 IF G>122 GOTO 410
460 IF G>90 GOTO 480
465 B=G
470 GOTO 490
480 B=G-32

485 ' *** Set first key pressed ***

490 IF FLAG=1 THEN F1=B:FLAG=2:PRINTTAB(4)"First key pressed
:";CHR$(KBD(B-64)):GOTO 410

495 ' *** Wrong Guess ***

500 PRINTTAB(4)"Second key pressed:";CHR$(KBD(B-64))
510 IF F1<>KBD(B-64) THEN PRINTTAB(4)"Sorry.  Wrong guess."
    :GOTO
600

515 ' *** Right Guess ***

520 KBD(B-64)=KBD(F1-64)
530 KBD(F1-64)=F1
540 C$=C$+CHR$(F1)
550 PRINTTAB(4)"Correct guess!!"
560 SC(PLAYER)=SC(PLAYER)+1
570 IF SC(PLAYER)>4 GOTO 630
580 PRINTTAB(4)"Letters guessed:";C$
590 IF PLAYER=1 THEN PLAYER=2 ELSE PLAYER=1
600 PRINT:PRINTTAB(9)"== Hit any key ==";
610 A$=INKEY$:IF A$=""GOTO 610
620 FLAG=1:GOTO 380

625 ' *** Winner ***

630 CLS:PRINT:PRINTTAB(6);
640 PRINT PLAYER$(PLAYER);" wins!!"
650 PRINT:PRINTTAB(8)"Play again?"
660 A$=INKEY$:IF A$="" GOTO 660
670 IF A$="Y" OR A$="y" THEN RUN
680 CLS
```

Chapter 23



Slot Machine

I have resisted the temptation to include Blackjack in this book, mainly because 21 or some other variation of the game is furnished free by most computer manufacturers, and I figure such games are worth what we pay for them. I got Blackjack free with my first Radio Shack computer in 1979 and another version with a VIC-20. If you want Blackjack for your Model 100, however, you are going to have to go out and pay dearly for it. This book gives you nothing more than Craps and, in this chapter, Slot Machine.

Most humans find slots pretty boring. I marvel at those who can stand at the real machine pumping money into them one coin at a time. I think humanity would be served if the makers of gambling machines would simply place a funnel on top, so that rolls of coins could be poured in all at once, and the tedium dispensed with. However, there is a certain element of humanity that is fascinated by slot machines. You know who you are. This program was written especially for you.

The player starts with $200, and may insert silver dollars one at a time. $20 is won each time three lemons, cherries, or oranges appear. $5 is awarded for hitting one of each. All other combinations lose. The object of the game is to see if you can keep hitting the keys long enough to lose all your money. I have rigged the odds so that your batteries will probably not hold out that long. Keep your AC adapter handy.

Printing the appropriate fruit to the screen is done by selecting a random number between one and three. As the number is chosen, the program goes to one of three routines, each of which prints a different fruit and increments the appropriate counter: C for cherries, L for lemons, or O1 for oranges. After three have been selected, the program looks to see if any of these three variables equals three, or if all three equal one. In either case, control branches to one of the WIN routines, and the player's score is incremented.

Just in case, a bankrupt routine is included,

A$	Used in INKEY$ loop
C	Number of cherries
CASH	Player's money
CHER$	Graphics representation of cherries
DELAY	Delay loop
DU	Dummy variable used for RND(1)
F$	Print using format
I	Loop counter
L	Number of lemons
LEMN$	Graphics representation of lemons
N	Loop counter
O1	Number of oranges
OGE$	Graphics representation of oranges

Fig. 23-1. Variables used in Slot Machine.

beginning at line 660. You should be so lucky to lose nothing but your imaginary money with your Model 100! A list of the variables used in the program is shown in Fig. 23-1.

Program Listing

```
 10 ' *********
 20 ' *       *
 30 ' * Slots *
 40 ' *       *
 50 ' *********

 55' *** Set Random Start Point ***

 60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 70 DU=RND(1)
 80 NEXT N
 90 F$="$$####"
100 CASH=200
110 LEMN$="^^LEMON^^"
120 CHER$="%%CHERRY%%"
130 OGE$="**ORANGE**"

135 ' *** Instructions ***
```

```
140 CLS:PRINT
150 PRINTTAB(10)"Instructions?"
160 PRINT:PRINTTAB(14)"Y/N"
170 A$=INKEY$:IF A$=""GOTO 170
180 IF A$="Y" OR A$="y" GOTO 190 ELSE GOTO 250
190 CLS
200 PRINT
210 PRINTTAB(2)"Drop in silver dollars.  You win"
220 PRINTTAB(2)"$20 each time you get three lemons,"
230 PRINTTAB(2)"three cherries, or three oranges."
240 PRINTTAB(2)"One of each wins $5.  Others lose."
250 PRINTTAB(8)"You start with $200"
260 PRINT:PRINTTAB(9)"== Hit any key =="
270 A$=INKEY$:IF A$=""GOTO 270

275 ' *** Start Round ***

280 C=0:L=0:O1=0
290 CLS:PRINT:PRINT:PRINT
300 PRINTTAB(4);
310 FOR I=1 TO 3
320 FOR DELAY=1 TO 250
330 NEXT DELAY

335 ' *** Choose Fruit ***

340 N=INT(RND(1)*3)+1
350 ON N GOTO 360,390,420
360 PRINT CHERY$;" ";
370 C=C+1
380 GOTO 440
390 PRINT LEMN$;" ";
400 L=L+1
410 GOTO 440
420 PRINT OGE$;" ";
430 O1=O1+1
440 NEXT I
450 FOR DELAY=1 TO 150
460 NEXT DELAY
470 IF C=3 OR L=3 OR O1=3 GOTO 590
480 IF C=1 AND L=1 AND O1=1 GOTO 500
```

```
490 GOTO 530

495 ' *** Won $5.00 ***

500 PRINT:PRINTTAB(9)"You won $5.00!!"
510 CASH=CASH+5
520 GOTO 610

525 ' *** Lost Bet ***

530 PRINT:PRINT:PRINT
540 PRINTTAB(6)"Oops. You lost your dollar"
550 CASH=CASH-1
560 PRINT
570 IF CASH=0 GOTO 660
580 GOTO 610

585 ' *** Won $20.00 ***

590 PRINT:PRINTTAB(9)"You won $20.00!!"
600 CASH=CASH+10
610 PRINTTAB(9)"You now have";
620 PRINT USING F$;CASH
630 PRINTTAB(9)"== Hit any key =="
640 A$=INKEY$:IF A$=""GOTO 640
650 GOTO 280

655 ' *** Went Bankrupt ***

660 CLS:PRINT
670 PRINTTAB(10)"You are bankrupt!"
680 PRINTTAB(10)"Sorry about that."
690 PRINT
700 PRINTTAB(12)"Another game?"
710 PRINT
720 PRINTTAB(16)"Y/N"
730 A$=INKEY$:IF A$=""GOTO 730
740 IF A$="Y" OR A$="y" THEN RUN
750 CLS
```

Chapter 24

Art Auction

Have you ever felt that you were at the mercy of a random universe? If so, you will like Art Auction. You get to buy paintings, bidding against an opponent who bids randomly. Then, you may sell your paintings, but all the offers will be random amounts, and even the number of offers that you will get is decided at random. Is that lacking in causality enough for you?

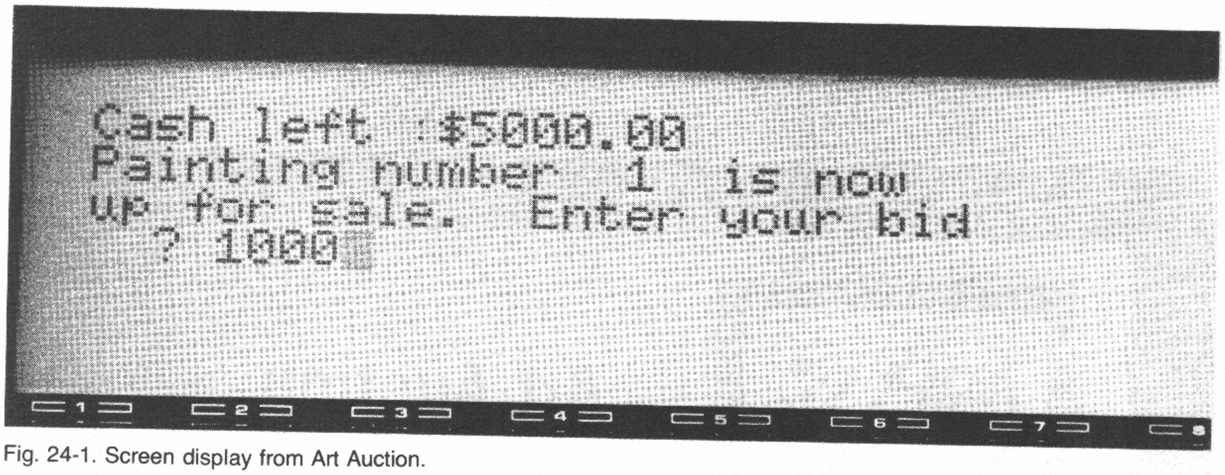
The player will start off with $5000 and be offered five different paintings. The object is to accumulate as many paintings as possible for the lowest price possible. Figure 24-1 shows the screen display during the first round of play. After you submit your bid, the computer will make a bid. If yours is higher, you get the painting. If your bid is a lot higher, you still get the art work, but you get the sick feeling of having spent too much money. This is very like real auctions in many ways.

In the second round of play, you will be offered one or more bids on your art. Some may be more than what you paid. Some may be lower. If you refuse a bid, and it turns out to be the last one, you are stuck with the painting. The object is to finish the game with more money than you started with. Thus, buying many paintings and selling them all for a profit is to be desired.

Your BID, which cannot be greater than your CASH, is entered in line 290. The opposing bid is randomly selected. A random number from 1 to 99 is chosen by the computer, multiplied by 10, and added to 150. So, the lowest bid the computer is likely to make will be $160. The highest it can bid is $1140. Knowing these facts will likely spoil the game for you, so forget you read them.

The amount you paid for a painting is stored in COLLECT(n), and your CASH reduced each time you successfully acquire a fine work of art.

In the second portion of the game, you are offered a number of bids, from one to seven, determined by a random number, OF, selected in line 510. Then, a for-next loop from 1 to OF bids anywhere from $600 to $2500 for your paintings. It's up



```
Cash left :$5000.00
Painting number  1  is now
up for sale.  Enter your bid
  ? 1000
```

Fig. 24-1. Screen display from Art Auction.

to you to accept or reject an offer. Strategy: anything over $2000 should be snapped up. Also: don't pay more than $2500 for a painting; you'll never get your investment back.

A list of the variables used in the program is shown in Fig. 24-2.

A$	Used in INKEY$ loop
BID	Amount bid for painting
CASH	Player's cash remaining
COLLECT(n)	Prices paid for up to five paintings
DU	Dummy variable for RND(1)
F$	Print using format
N	Loop counter
N2	Loop counter
OF	Number of offers to be made
OP	Opponent's bid

Fig. 24-2. Variables used in Art Auction.

Program Listing

```
10 ' ***************
20 ' *             *
30 ' * Art Auction *
40 ' *             *
50 ' ***************
```

```
55 ' *** Set Random Start Point ***

60 FOR N=1 TO VAL(RIGHT$(TIME$,2))
70 DU=RND(1)
80 NEXT N

85 ' *** Instructions ***

 90 CLS:PRINT:PRINT
100 PRINTTAB(12)"Instructions?"
110 PRINT:PRINTTAB(16)"(Y/N)"
120 A$=INKEY$:IF A$="" GOTO 120
130 IF A$="Y" OR A$="y" GOTO 140 ELSE GOTO210
140 CLS:PRINT
150 PRINTTAB(7)"You will get to buy up"
160 PRINTTAB(7)"to five paintings.  Then"
170 PRINTTAB(7)"they will be auctioned."
180 PRINTTAB(7)"Number of offers will vary."
190 PRINT:PRINTTAB(9)"== Hit any key ==";
200 A$=INKEY$:IF A$="" GOTO 200
210 CASH=5000
220 F$="$$###.##"

225 ' *** Buy Paintings ***

230 FOR N=1 TO 5
240 CLS:PRINT
250 PRINTTAB(2)"Cash left :";USING F$;CASH
260 PRINTTAB(2)"Painting number ";N;" is now"
270 PRINTTAB(2)"up for sale.  Enter your bid"
280 PRINTTAB(4);
290 INPUT BID$
300 BID=VAL(BID$)
310 IF BID>CASH GOTO 240
320 OP=INT(RND(1)*100)*10+150
330 PRINTTAB(2)"Opponent bid ";
340 PRINT USING F$;OP
350 IF BID=>OP GOTO 380
360 PRINTTAB(2)"You lost that one!"
370 GOTO 420
380 PRINTTAB(2)"You bought that one!"
```

```
390 NU=NU+1
400 COLLECT(NU)=BID
410 CASH=CASH-BID
420 PRINTTAB(2)"== Hit any key ==";
430 A$=INKEY$:IF A$=""GOTO 420
440 NEXT N

445 ' *** Sell  Paintings ***

450 FOR N=1 TO NU
460 CLS:PRINT
470 PRINTTAB(2)"Your painting #";N;"now for"
480 PRINTTAB(2)"sale. You paid ";
490 PRINT USING F$;COLLECT(N);
500 PRINTTAB(2)"."
510 OF=INT(RND(1)*6)+1
520 FOR N2=1 TO OF
530 BID=INT(RND(1)*200)*10+500
540 PRINTTAB(2)"Bid :";
550 PRINT USINGF$;BID
560 PRINT:PRINTTAB(2)"Accept (Y/N)"
570 A$=INKEY$:IF A$=""GOTO 570
580 IF A$="Y" OR A$="y" GOTO 590 ELSE GOTO 610
590 CASH=CASH+BID
600 GOTO 670
610 NEXT N2

615 ' *** Too late! ***

620 CLS:PRINT
630 PRINTTAB(2)"Sorry, last offer!"
640 PRINTTAB(2)"You're stuck with it!"
650 PRINT:PRINTTAB(10)"== Hit any key ==";
660 A$=INKEY$:IF A$="" GOTO 660
670 NEXT N

675 ' *** End of Game ***

680 CLS:PRINT
690 PRINTTAB(2)"You started with $5000"
700 PRINTTAB(2)"And finish with ";
710 PRINT USING F$;CASH
```

```
720 IF CASH<5000 THEN PRINT:PRINT"Sorry about that!":GOTO 740
730 PRINT:PRINTTAB(10)"Good trading!"
740 PRINT:PRINTTAB(10)"Another game?"
750 A$=INKEY$:IF A$=""GOTO 750
760 IF A$="Y" OR A$="y" THEN RUN
770 CLS
780 END
```

Chapter 25

THOUSANDS OF SATED ELM TREES SWIM SILENTLY

Memory Stretcher

Test your ability to remember things displayed only briefly. Memory Stretcher shows you a sentence assembled from a random collection of adjectives, adverbs, nouns, and verbs by the computer. The sentence will be grammatically correct, but it may not mean much. A sample sentence is shown in Fig. 25-1. You will be shown the sentence only for a few seconds or for a fraction of a second as you improve. You will be required to type out the sentence exactly after it has vanished from the screen.

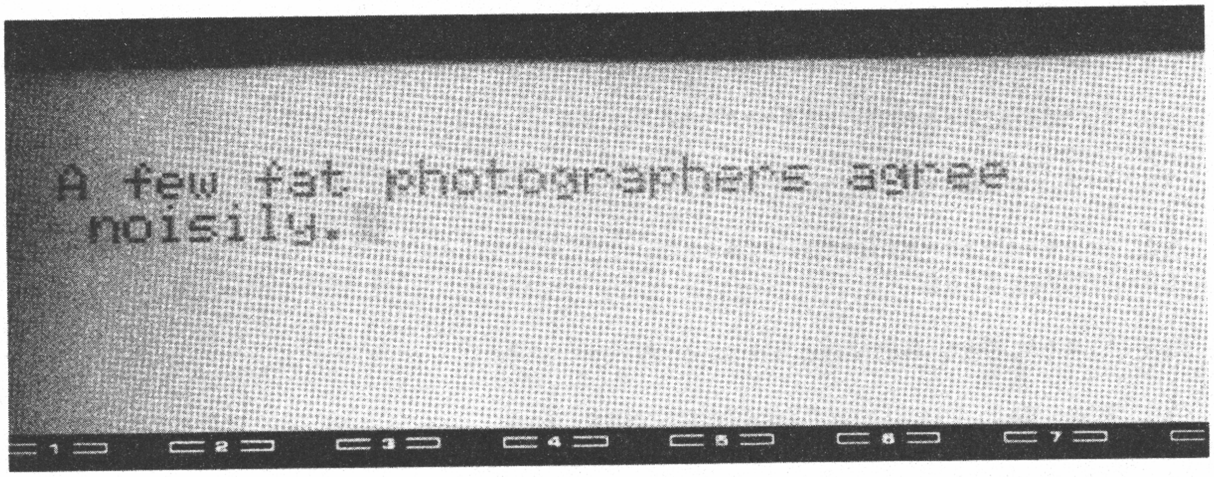
If you do so, the computer will show you another sentence, for a briefer time period. This continues until either you or the computer reaches the limit.

The sentences are made up of five words or phrases, stored in data lines and loaded into an array, S$(row,column). Twenty of each word type are available, producing thousands of different sentence combinations. The computer will select one segment from each column and then print it to the screen.

The length of time the sentence appears on the screen is determined by a delay loop in line 630. The loop at first repeats 500 times. If the player types in the sentence correctly, the delay is decreased by 25 percent. If the player is wrong, the delay is increased by 25 percent.

To make the sentence easier to read, it is not split in the middle of words. The individual letters of the sentence are printed to the screen, one character at a time, in a for-next loop from 1 to LEN(S$). As soon as the sentence is 25 characters long, FLAG is set to one, and henceforth, the program looks at MID$(S$,N,1) to see if that character is a space. If it is, a carriage return is used to move the next word down one line.

The player's answer, G$, is compared with the sentence and has to match exactly. If it does not, control passes to line 690, where the delay is lengthened. Also, the variable LOSS is incremented. If LOSS ever equals 2, the player has missed two in a row, and the game is over.



```
A few fat photographers agree
 noisily.
```

Fig. 25-1. Screen display from Memory Stretcher.

The delay loop is shortened in the correct routine beginning at line 950. LOSS is returned to zero to give the player two more misses before the game is over.

At the end of play, the participant is told what level of proficiency has been reached. This is determined by dividing the length of the delay at the end of the game by 10.

A list of the variables used in the program is shown in Fig. 25-2.

A$	Used in INKEY$ loop
COLUMN	Column of sentence data
DELAY	Delay on screen
DU	Dummy variable for RND(1)
FLAG	Set when sentence longer than 25 characters
G$	Player version of sentence displayed
LE	Level reached by player
LOSS	Number of sentences in a row missed
N	Loop counter
R	Random number
ROW	Row of sentence data
S$	Assembled sentence

Fig. 25-2. Variables used in Memory Stretcher.

Program Listing

```
 10 ' *********************
 20 ' *                   *
 30 ' *  Memory Stretcher *
 40 ' *                   *
 50 ' *********************
 60 CLEAR 1000

 55 ' *** Set Random Start Point ***

 70 FOR N=1 TO VAL(RIGHT$(TIME$,2))
 80 DU=RND(1)
 90 NEXT N
100 DIM S$(20,5)
110 DELAY=500
120 DATA Several,Some,Many,A few,Thousands of,These,Those
130 DATA Your,My,His,Her,America's,No doubt,Frequently
140 DATA Sometimes,Occasionally,Yesterday,Today,
         Always,Once a month
150 DATA yellow,red,fat,large,small,big,thin,hungry,sated
160 DATA predicted,early,late,appreciated,smart,stupid
170 DATA enthusiastic,rowdy,polite,common,silly
180 DATA politicians,programmers,elephants,students,swimmers
190 DATA air traffic controllers,garbage cans,elm trees,
         German Shepherds,photographers
200 DATA mice,cheap-skates,hoboes,strangers,relatives
210 DATA samurai,goats,submarines,cockroaches,writers
220 DATA correspond,vegetate,select,interpret,orate
230 DATA predict,inspire,slaughter,sleep,swim
240 DATA bark,shout,arrive,depart,investigate
250 DATA agree,dispair,perambulate,instigate,perspire
260 DATA noisily,noisomely,annoyingly,cloyingly,loudly
270 DATA softly,angrily,eagerly,silently,predictably
280 DATA inanely,quickly,stealthily,competently,
         mysteriously
290 DATA regularly,enthusiastically,selectively,
         incredibly,partially

295 ' *** Instructions ***
```

```
300 CLS:PRINT:PRINT
310 PRINTTAB(12)"Instructions?"
320 PRINT
330 PRINTTAB(16)"Y/N"
340 A$=INKEY$:IF A$="" GOTO 340
350 IF A$="Y" OR  A$="y" GOTO 360 ELSE GOTO 410
360 CLS:PRINT
370 PRINTTAB(2)"Try to remember the phrase, and"
380 PRINTTAB(2)"type it in.  It appears on screen"
390 PRINTTAB(2)"for shorter time each right guess."
400 PRINTTAB(2)"Two wrong guesses in row ends game."

405 ' *** Read Sentence Data ***

410 FOR COLUMN=1 TO 5
420 FOR ROW=1 TO 20
430 READ S$(ROW,COLUMN)
440 NEXT ROW
450 NEXT COLUMN
460 PRINT
470 PRINTTAB(9)"== Hit any key =="
480 A$=INKEY$:IF A$="" GOTO 480
490 IF LOSS=2 GOTO 1070

495 ' *** Assemble Sentence ***

500 FOR N=1 TO 5
510 R=INT(RND(1)*20)+1
520 S$=S$+S$(R,N)+CHR$(32)
530 NEXT N
540 S$=LEFT$(S$,LEN(S$)-1)
550 S$=S$+"."

555 ' *** Flash on Screen ***

560 CLS:PRINT:PRINT
570 PRINTTAB(2);
580 FOR N=1 TO LEN(S$)
590 IF FLAG=1 GOTO 610
600 IF N>25 AND MID$(S$,N,1)=CHR$(32) THEN
    PRINT:PRINTTAB(2);:FLAG=1
```

```
610 PRINT MID$(S$,N,1);
620 NEXT N
630  FOR N=1 TO DELAY:NEXT N
640 FLAG=0

645 ' *** Player Re-Enters ***

650 CLS
660 PRINT"Re-enter the sentence:"
670 LINEINPUT G$
680 IF G$=S$ GOTO 950

685 ' *** Player Wrong ***

690 CLS:PRINT
700 PRINTTAB(2)"Wrong-o!"
710 PRINTTAB(2)"You had:"
720 PRINTTAB(1)"";
730 FOR N=1 TO LEN(G$)
740 IF FLAG=1 GOTO 760
750 IF N>25 AND MID$(G$,N,1)=CHR$(32) THEN
    PRINT:PRINTTAB(2);:FLAG=1
760 PRINT MID$(G$,N,1);
770 NEXT N
780 PRINT
790 FLAG=0
800 PRINTTAB(2)"Correct sentence:"
810 PRINTTAB(1)"";
820 FOR N=1 TO LEN(S$)
830 IF FLAG=1 GOTO 850
840 IF N>25 AND MID$(S$,N,1)=CHR$(32) THEN
    PRINT:PRINTTAB(2);:FLAG=1
850 PRINT MID$(S$,N,1);
860 NEXT N
870 PRINT:PRINTTAB(2)"We'll make it easier next time."
880 DELAY=DELAY*1.25
890 FLAG=0:S$="":G$=""
900 PRINT
910 PRINTTAB(9)"== Hit any key =="
920 A$=INKEY$:IF A$="" GOTO 920
930 LOSS=LOSS+1
```

```
 940 GOTO 490
 945 ' *** Player Right ***
 950 CLS:PRINT
 960 PRINTTAB(12)"Right!"
 970 PRINT
 980 PRINTTAB(2)"Now let's make it harder!"
 990 DELAY=DELAY*.75
1000 FLAG=0
1010 G$="":S$=""
1020 PRINT
1030 PRINTTAB(9)"== Hit any key =="
1040 A$=INKEY$:IF A$="" GOTO 1040
1050 LOSS=0
1060 GOTO 490

1065 '  *** Game Over ***

1070 CLS:PRINT:PRINT
1080 PRINTTAB(2)"Sorry, you missed two in a row."
1090 LE=INT(DELAY/10)
1100 PRINTTAB(2)"However, you did reach Level";LE
1110 PRINT
1120 PRINTTAB(8)"Play again?"
1130 PRINTTAB(12)"Y/N"
1140 A$=INKEY$:IF A$="" GOTO 1140
1150 IF A$="Y" OR A$="y" THEN RUN
```

Chapter 26

Reflex

Because computers operate so quickly, they are good tools to use for things that happen very fast. Your Model 100, for example, is an excellent choice for tracking the depletion of your checking or savings account. In the games realm, it is useful for pitting one human against another. It cannot lie nor show favoritism. If the Model 100 says that Player One has won the game, you can be certain that this is so, as long as Player One is not also the programmer.

Reflex is more secure, having been written by me. Two players (dubbed Player Right and Player Left to avoid confusion) test their reflexes against each other. Player Right should stand or sit on the right side of the computer, while Player Left stands or sits on the left. Again, this is to avoid confusion and is optional for nonconformists.

When the computer displays a signal an enthusiastic NOW!!! flashed on the screen, each player swats either the ESC key, which is at the upper left-hand side of the keyboard, or the DEL key, which is at the upper right-hand side. You may deduce which player hits which key.

The computer judges which player was first and keeps track. Whichever player accumulates nine points first wins. Jumping the gun is frowned upon and results in a one point penalty. You math types can think of a penalty as a negative number, which decrements your score.

In operation, the computer sets a delay, which is a random number between 500 and 2500, and then loops through a for-next loop from 1 to DELAY. If a player happens to press a key during this delay, it is detected in line 300 and control branches to 580, which is the jumped-the-gun routine. An appropriate scolding is printed to the screen, and that player's point total is diminished by one.

If all goes well during the delay loop, the NOW!!! will be printed to the screen and control will drop down to another INKEY$ loop, beginning at 330. An irritating sound is supplied while the computer awaits keyboard input. If the ESC key or

A$	Used in INKEY$ loop
AN$	Player answer
DELAY	Length of delay loop
DU	Dummy variable for RND(1)
N	Loop counter
PL	Score of Player Left
PR	Score of Player Right

Fig. 26-1. Variables used in Reflex.

DEL key is pressed, the program branches to line 390 to reward the winner. Otherwise, the key depression is ignored. The program makes someone press the correct key.

Once one of the players has struck one of the acceptable keys, the winning player is notified in routines at lines 400 and 490, his or her score is upped by one, and control is returned to line 260 so that the players can play again. If the score of Player Right or Left exceeds nine (line 270), a winner is declared and the game is over.

A list of the variables used in the program is shown in Fig. 26-1.

Program Listing

```
 10 ' **********
 20 ' *        *
 30 ' * Reflex *
 40 ' *        *
 50 ' **********

 55 ' *** Instructions ***

 60 CLS:PRINT:PRINT
 70 PRINTTAB(11)"*****************"
 80 PRINTTAB(11)"*    Reflex     *"
 90 PRINTTAB(11)"*****************"
100 PRINTTAB(13)"Instructions?"
110 PRINT:PRINTTAB(17)"Y/N"
120 A$=INKEY$:IF A$=""GOTO120
130 IF A$="Y" OR A$="y" GOTO150
140 GOTO230
150 CLS:PRINT
160 PRINTTAB(2)"When given the signal, player on"
170 PRINTTAB(2)"right will attempt to hit
    ";CHR$(34)"DEL";CHR$(34)
180 PRINTTAB(2)"key before player on the left hits"
190 PRINTTAB(2)"the ";CHR$(34);"ESC";CHR$(34);" key.
    First to reach 10"
200 PRINTTAB(2)"points wins.":PRINT
```

```
210 PRINTTAB(10)"== Hit any key =="
220 A$=INKEY$:IF A$="" GOTO220

225 ' *** Set random start point ***

230 FOR N=1 TO VAL(RIGHT$(TIME$,2))
240 DU=RND(1)
250 NEXT N
260 CLS
270 IF PL>9 OR PR>9 GOTO710

275 ' *** Count off random delay ***

280 DELAY=RND(1)*2000+500
290 FOR N=1 TO DELAY
300 A$=INKEY$:IF A$<>""GOTO580
310 NEXT N
320 PRINT@175,"N O W !!!!"

325 ' *** Look for ESC or DEL key ***

330 A$=INKEY$
340 SOUND 10000,1
350 IF A$=""GOTO330
360 A=ASC(A$)
370 IF A=27 OR A=8 GOTO390
380 GOTO330
390 IF A=27 GOTO490

395 ' *** Player Right Won ***

400 CLS
410 PR=PR+1
420 PRINTTAB(4)"Points Left";TAB(26)"Points Right"
430 PRINTTAB(6);PL;TAB(30);PR
440 PRINT@180,"Winner : =======>"
450 PRINT
460 PRINTTAB(8)" == Hit any key =="
470 A$=INKEY$:IF A$=""GOTO470
480 GOTO260

485 ' *** Player Left Won ***
```

```
490 CLS
500 PL=PL+1
510 PRINTTAB(4)"Points Left";TAB(26)"Points Right"
520 PRINT:PRINTTAB(6);PL;TAB(30);PR
530 PRINT:PRINT"<====== Winner"
540 PRINT
550 PRINTTAB(10)"== Hit any key =="
560 A$=INKEY$:IF A$=""GOTO560
570 GOTO260

575 ' *** Somebody Jumped Gun ***

580 A=ASC(A$)
590 IF A=27 OR A=8 GOTO610
600 GOTO310
610 IF A=27 THEN PL=PL-1 ELSE GOTO650
620 PRINTTAB(4)"Player Left Jumped the gun!!"
630 PRINTTAB(4)"You lose one point, sport."
640 GOTO680
650 PR=PR-1
660 PRINTTAB(4)"Player Right Jumped the gun!!"
670 PRINTTAB(4)"You lose one point, sport."
680 PRINT:PRINTTAB(10)"== Hit any key =="
690 A$=INKEY$:IF A$=""GOTO690
700 GOTO260

705 ' *** Winners Declared ***

710 CLS:PRINT
720 IF PL>9 THEN PRINTTAB(6)"Player Left wins!":GOTO740
730 PRINTTAB(8)"Player Right wins!"
740 PRINT:PRINT
750 PRINTTAB(8)"Play again";
760 INPUT AN$
770 IF LEFT$(AN$,1)="Y" OR LEFT$(AN$,1)="y" THEN RUN
```

Index

Index